软件工程师培养丛书

工信部国家级计算机人才评定体系

ASP.NET网站开发

武汉厚溥教育科技有限公司　编著



"理论→总结→上机→习题"四阶段教学模式

- 理论结合实践，注重动手能力培养
- 任务驱动讲解，有效激发学习兴趣
- 典型项目案例，扎实培养专业素质
- 教学做一体化，极大提高教学效率

清华大学出版社

软件工程师培养丛书

工信部国家级计算机人才评定体系

ASP.NET 网站开发

武汉厚溥教育科技有限公司 编著

清华大学出版社

北 京

内 容 简 介

本书按照高等院校、高职高专计算机课程基本要求，以案例驱动的形式来组织内容，突出计算机课程的实践性特点。本书共包括 7 章：LINQ to SQL、用户控件与 HttpHandler、成员资格和角色管理、个性化用户配置、数据缓存、母版页与站点导航、项目整合和主题。

本书附赠 PPT 教学课件和相关教辅资料，这些教学资源可通过 http://www.tupwk.com.cn 或 http://www.hop-e.net下载。

本书内容安排合理，层次清楚，通俗易懂，实例丰富，突出理论和实践的结合，可作为各类高等院校、高职高专及培训机构的教材，也可供广大程序设计人员参考。

图书在版编目(CIP)数据

ASP.NET 网站开发/武汉厚溥教育科技有限公司编著.—北京：清华大学出版社，2016
(软件工程师培养丛书)
ISBN 978-7-302-42410-9

Ⅰ. ①A… Ⅱ. ①武… Ⅲ. ①网页制作工具—程序设计 Ⅳ. ①TP393.092

中国版本图书馆 CIP 数据核字(2015)第 298590 号

责任编辑：刘金喜　蔡　娟
封面设计：崔东方
版式设计：妙思品位
责任校对：成风进
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课 件 下 载：http://www.tup.com.cn，010-62794504
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张：**20.25　　**字　　数：**341 千字
版　　次：2016 年 1 月第 1 版　　**印　　次：**2016 年 1 月第 1 次印刷
印　　数：1～2500
定　　价：39.80 元

产品编号：

编 委 会

前　言

ASP.NET 的前身 ASP 技术，是在 IIS 2.0(Windows NT 3.51)上首次推出的，当时与 ADO 1.0 一起推出，在 IIS 3.0(Windows NT 4.0)上发扬光大，成为服务器端应用程序的热门开发工具，微软公司还特别为它量身打造了 Visual InterDev 开发工具。在 1994 年到 2000 年之间，ASP 技术已经成为微软推广 Windows NT 4.0 平台的关键技术之一，数以万计的 ASP 网站也是这个时候开始如雨后春笋般地出现在网络上，它的简单以及高度可定制化的能力，也是它能迅速崛起的原因之一。

ASP.NET 是.NET FrameWork 的一部分，是一项微软公司的技术，是一种使嵌入网页中的脚本可由因特网服务器执行的服务器端脚本技术，它可以在通过 HTTP 请求文档时再在 Web 服务器上动态创建它们。ASP 是指 Active Server Pages(动态服务器页面)，是运行于 IIS(Internet Information Server，是 Windows 开发的 Web 服务器)之中的程序。

本书是“软件工程师培养丛书”中的一本专业教材，该丛书是由武汉厚溥教育科技有限公司开发，以培养符合企业需求的软件工程师应用开发、实施为目标的 IT 职业教育丛书。在开发该丛书之前，我们对 IT 行业的岗位序列做了充分的调研，包括研究从业人员技术方向、项目经验和职业素质等方面的需求，通过对面向的学生的特点、行业需求的现状以及实施等方面的详细分析，结合“厚溥”对软件人才培养模式的认知，按照软件专业总体定位要求，进行软件专业产品课程体系设计。该丛书集应用软件知识和多领域的实践项目于一体，着重培养学生的熟练度、规范性、集成和项目能力，从而达到预定的培养目标。



本书共包括 7 章：LINQ to SQL、用户控件与 HttpHandler、成员资格和角色管理、个性化用户配置、数据缓存、母版页与站点导航、项目整合和主题。

我们对本书的编写体系做了精心的设计，按照“理论学习—知识总结—上机操作—课后习题”这一思路进行编排。“理论学习”部分描述通过本案例要达到的学习目标与涉及的相关知识点，使学习目标更加明确；“知识总结”部分概括案例所涉及的知识点，使知识点完整系统地呈现；“上机操作”部分对案例进行了详尽分析，通过完整的步骤帮助读者快速掌握该案例的操作方法；“课后习题”部分帮助读者理解章节的知识点。本书在内容编写方面，力求细致全面；在文字叙述方面，注意言简意赅、重点突出；在案例选取方面，强调案例的针对性和实用性。

本书凝聚了编者多年来的教学经验和成果，可作为各类高等院校、高职高专及培训机构的教材，也可供广大程序设计人员参考。

本书附赠 PPT 教学课件和相关教辅资料，这些教学资源可通过 http://www. tupwk.com.cn 或 http://www. hop-e.net 下载。

本书由武汉厚溥教育科技有限公司编著，由翁高飞、刘伟等多名企业实战项目经理编写。本书编者长期从事项目开发和教学实施，并且对当前高校的教学情况非常熟悉，在编写过程中充分考虑到不同学生的特点和需求，加强了项目实战方面的教学。本书编写过程中，得到了武汉厚溥教育科技有限公司各级领导的大力支持，在此对他们表示衷心的感谢。

参与本书编写的人员还有：武汉商学院教师曹静、荆州职业技术学院教师方风波、武汉工程职业技术学院教师邹治伟和彭莉、湖北三峡职业技术学院教师李建利、武汉软件工程职业学院教师谢日星、黄冈职业技术学院教师吴金秀、湖北国土资源职业学院教师徐霞和明素华等。

限于编写时间和编者的水平，书中难免存在不足之处，希望广大读者批评指正。

服务邮箱：wkservice@163.com　hop-e@foxmail.com。

编　者

2015 年 10 月

目　　录



第1章

LINQ to SQL



课程目标

- LINQ to SQL
- 动态数据支持



简　介

LINQ to SQL 是将对象关系映射到.NET 框架中的一种实现。它可以将关系数据库(如 SQL Server 数据库)映射为.NET Framework 中的一些类。然后，开发人员就可以使用 LINQ to SQL 对数据库中的数据进行查询、修改、插入、删除等操作。总之，通过使用 LINQ to SQL，开发人员可以使用 LINQ 技术访问基于关系的数据库，就如同访问内存中的集合一样。

1.1　LINQ to SQL 概述

LINQ to SQL 是 LINQ 中最重要的一个组件，为.NET Framework 3.5 所支持，它可以为关系数据库提供一个对象模型，并在该对象模型基础上实现对数据的查询、添加、修改、删除等功能，即 LINQ to SQL 提供了用于将关系数据作为对象管理的运行时基础结构。本章节介绍 LINQ to SQL 的基础知识，以及使用 LINQ to SQL 为 SQL Server 数据库创建对象模型的方法。

为了更加具体地说明 LINQ to SQL 的功能，以下使用 SQL Server 数据库来介绍 LINQ to SQL 的功能。

LINQ to SQL 最重要的一个功能就是为 SQL Server 数据库创建一个对象模型(由基于.NET 框架的类组成)，并将该对象模型映射到 SQL Server 数据库中相应的对象(如表、列、外键关系、存储过程、函数等)。其中，LINQ to SQL 类映射到 SQL Server 数据库中的表，这些 LINQ to SQL 类被称为“实体类”。实体类中的属性或字段映射到 SQL Server 数据库中表的列。实体类之间的关联映射到 SQL Server 数据库中的外键关系。LINQ to SQL 类中的方法映射为 SQL Server 数据库中的存储过程或函数。LINQ to SQL 对象模型和 SQL Server 数据库中的对象的映射关系如表 1-1 所示。



表 1-1　LINQ to SQL 对象模型和 SQL Server 数据库中的对象映射关系

LINQ to SQL 对象模型的基本元素	SQL Server 数据库中的对象
实体类	表
属性或字段	列
关联	外键关系
方法	存储过程或函数

下面使用 SQL Server 数据库 LinqDB 介绍 LINQ to SQL 对象模型和 SQL Server 数据库中的对象的映射关系。其中，LinqDB 数据库包含多个表，如 UserInfo、UserRole、Role 等。LinqDB 数据库中部分表(只列出了 UserInfo、UserRole 和 Role 表，其他表已经省略)的关系图如图 1-1 所示。

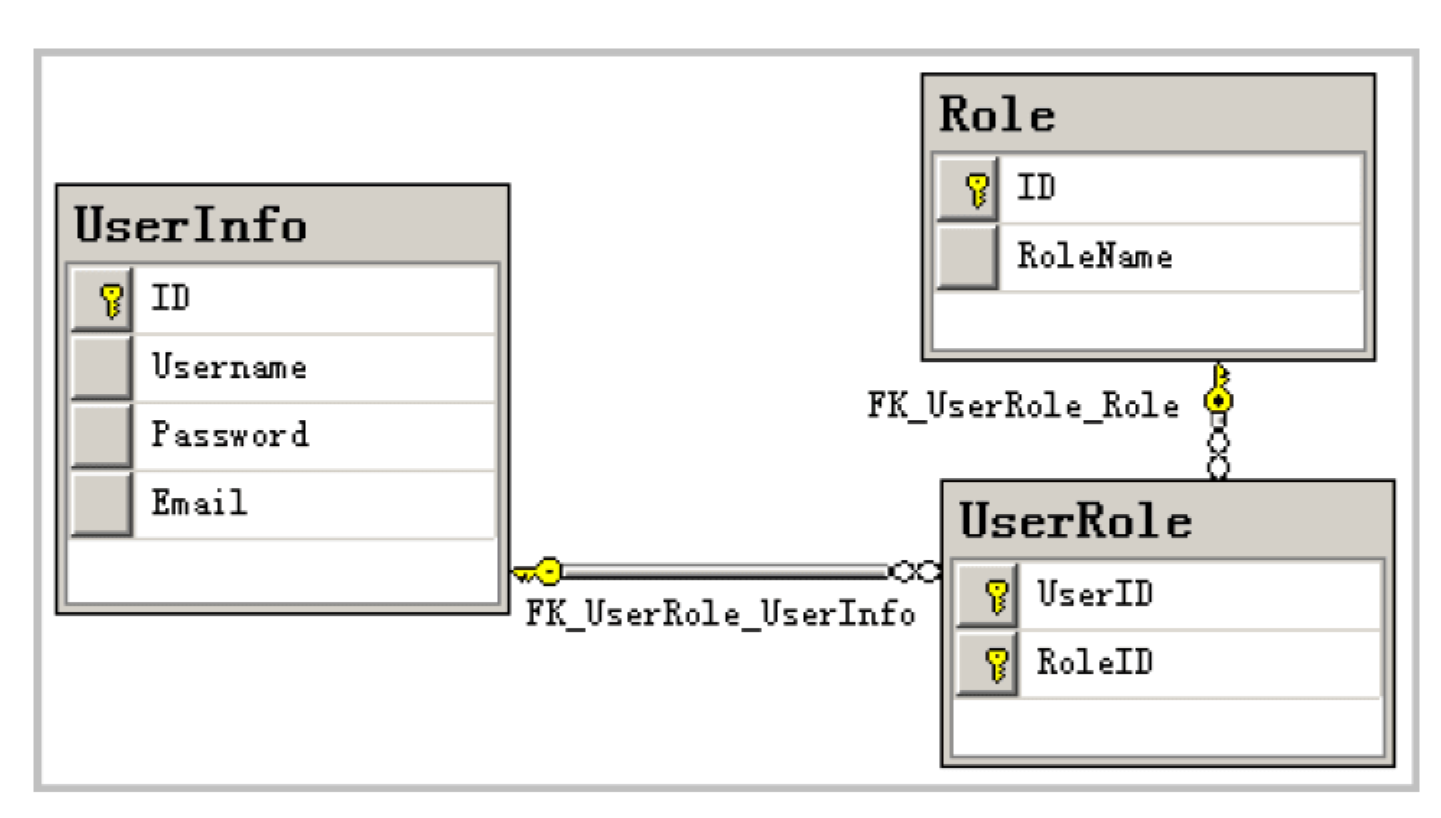


图 1-1

1．实体类和数据库表

实体类和.NET Framework 中的其他类比较相似。但是，实体类使用 TableAttribute 属性来描述其与数据库中表的关联关系。下面的实例代码创建了一个名称为“UserInfo”的实体类，该实体类映射到 LinqDB 数据库中的 UserInfo 表。

```
[Table(Name="UserInfo")]
public class UserInfo { ... }
```

2. 属性或(字段)和数据库表中的列

如果一个实体类映射到数据库中的表，那么它的属性或字段可以被映射到数据库表中的列。其中，它们之间的映射关系使用 ColumnAttribute 属性表示。下面的实例代码创建了一个名称为 UserInfo 的实体类，该实体类映射到 LinqDB 数据库中的 UserInfo 表。ID 属性映射到 UserInfo 表的 ID 列，并且 ID 列为 UserInfo 表的主键。Username 属性映射到 UserInfo 表中的 Username 列。

```
[Table(Name="UserInfo")]
public class UserInfo
{
    private int iD;
    private string username;
    /// <summary>
    /// 映射 ID 列
    /// </summary>
    [Column(IsPrimaryKey = true)]
    public int ID
    {
        get { return iD; }
        set { iD = value; }
    }
    /// <summary>
    /// 映射 Username 列
    /// </summary>
    [Column]
    public string Username
    {
```



```
        get { return username; }
        set { username = value; }
    }
    ...
}
```

3. 关联和数据库外键关系

在 LINQ to SQL 中，LinqDB 数据库中的外键关联通过 AssociationAttribute 属性表示。LinqDB 数据库中的 UserInfo 和 UserRole 表之间就存在外键关系(UserRole 表的 UserID 列应用 UserInfo 表的 ID 列作为外键)。下面的实例代码在 UserInfo 类中创建了一个名称为 UserRole 的关联，该关联映射到 UserInfo 和 UserRole 表的外键关系(UserInfo_UserRole)。

```
[Table(Name="UserInfo")]
public class UserInfo
{
     ...
    private EntitySet<UserRole> _UserRole;
    [Association(Name="UserInfo_UserRole",
        Storage="_UserRole", OtherKey="UserID")]
    private EntitySet<UserRole> UserRole
    {
        get { return _UserRole; }
        set { _UserRole.Assign(value); }
    }
    ...
}
```

1.2 使用 Visual Studio 2008 创建 DBML 文件

本节介绍使用 Visual Studio 2008 为 SQL Server 数据库 LinqDB 创建 DBML 文件的方法，并为该数据库中的表(如 UserInfo、Role、UserRole、Category 等)创建实体类，为每一个表的列创建相应的属性。如 LinqDB 数据库中的 UserInfo 表将和 LinqDB.dbml 文件中的 UserInfo 类进行映射，UserInfo 表的列和 UserInfo 表的属性进行映射，它们之间的映射关系如图 1-2 所示。

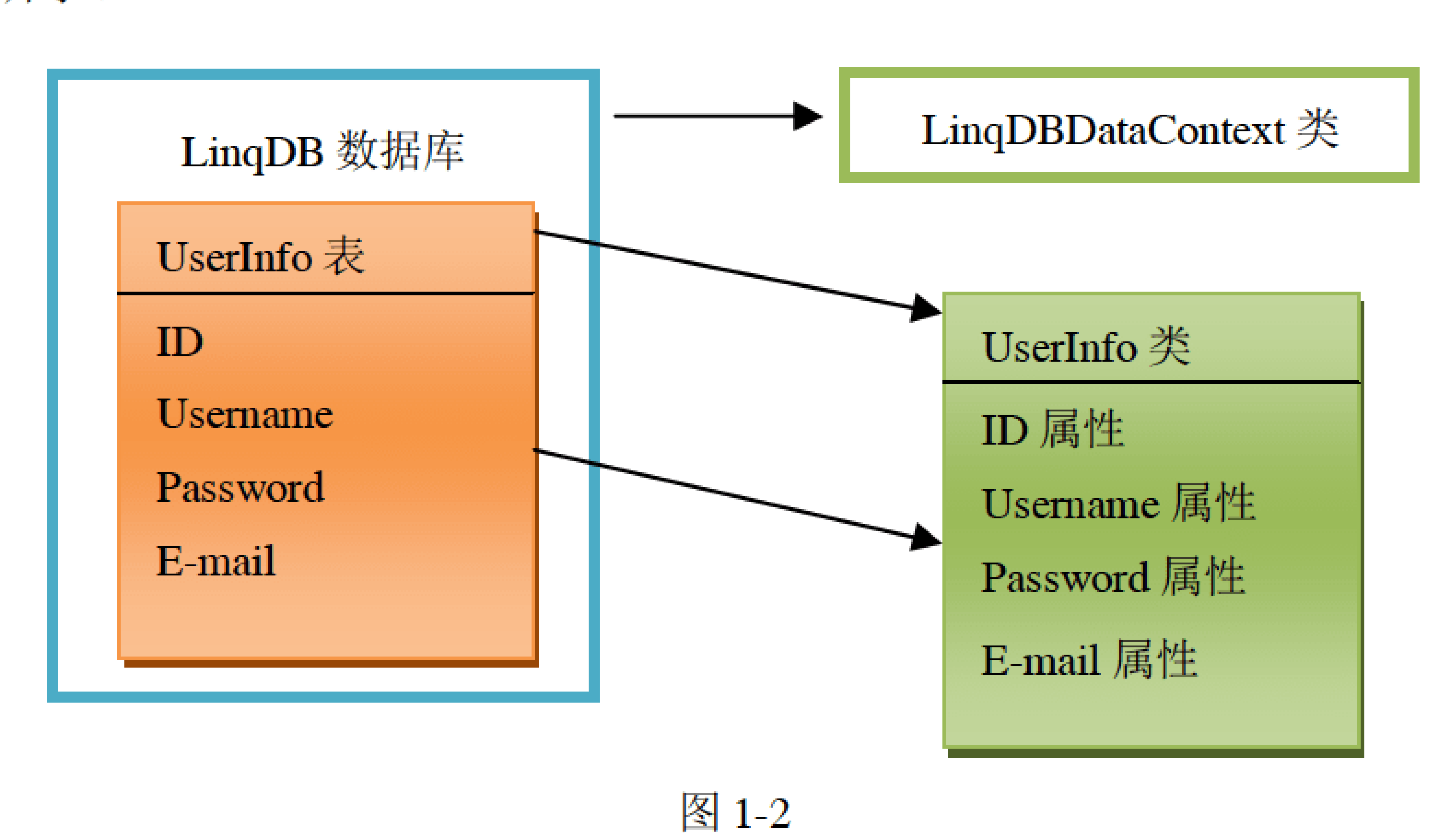


图 1-2

下面介绍在使用 Visual Studio 2008 为 SQL Server 数据库 LinqDB 创建 DBML 文件的方法，具体步骤如下。

(1) 在 Visual Studio 2008 集成开发环境中，右键单击“解决方案资源管理器”面板中的“App_Code”节点，并选择“添加新项”命令，操作如图 1-3 所示。



图 1-3

(2) 单击“添加新项”命令，弹出“添加新项”对话框。选择“LINQ to SQL 类”图标，并在“名称”输入框中输入“LinqDB.dbml”，如图 1-4 所示。



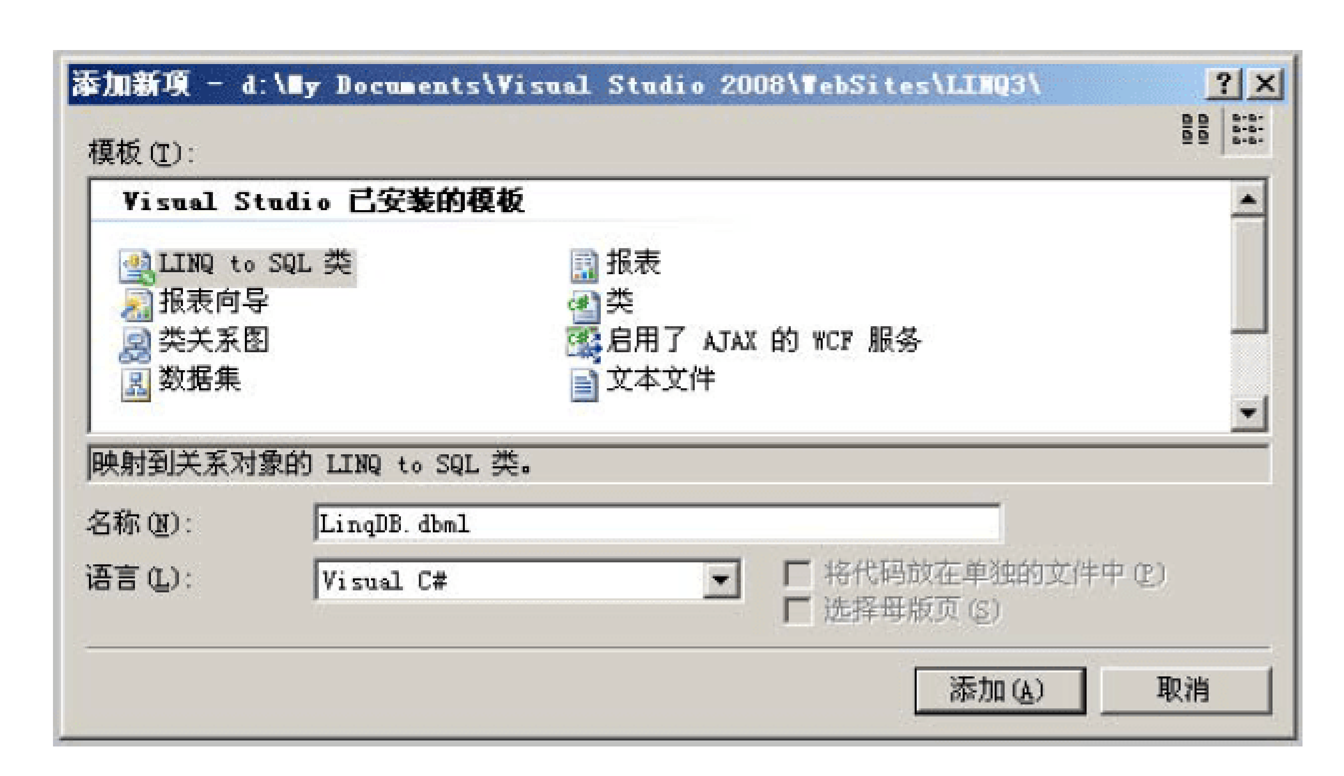


图 1-4

(3) 在“服务器资源管理器”面板中选择 LinqDB 数据库的各个表，并直接拖放到 LimlDB.dbml 文件的视图面板中。

(4) 将LinqDB数据库中所有的表和存储过程都直接拖放到LinqDB.dbml文件的视图面板中。

(5) 单击按钮保存上述操作的修改，即可创建 LinqDB.dbml 文件。

(6) 在“解决方案资源管理器”面板中查看 LinqDB.dbml 文件。

(7) 其中，LinqDB.dbml.1ayout 文件保存 LinqDB.dbml 文件的布局，该文件为一个 XML 文件，如图 1-5 所示。

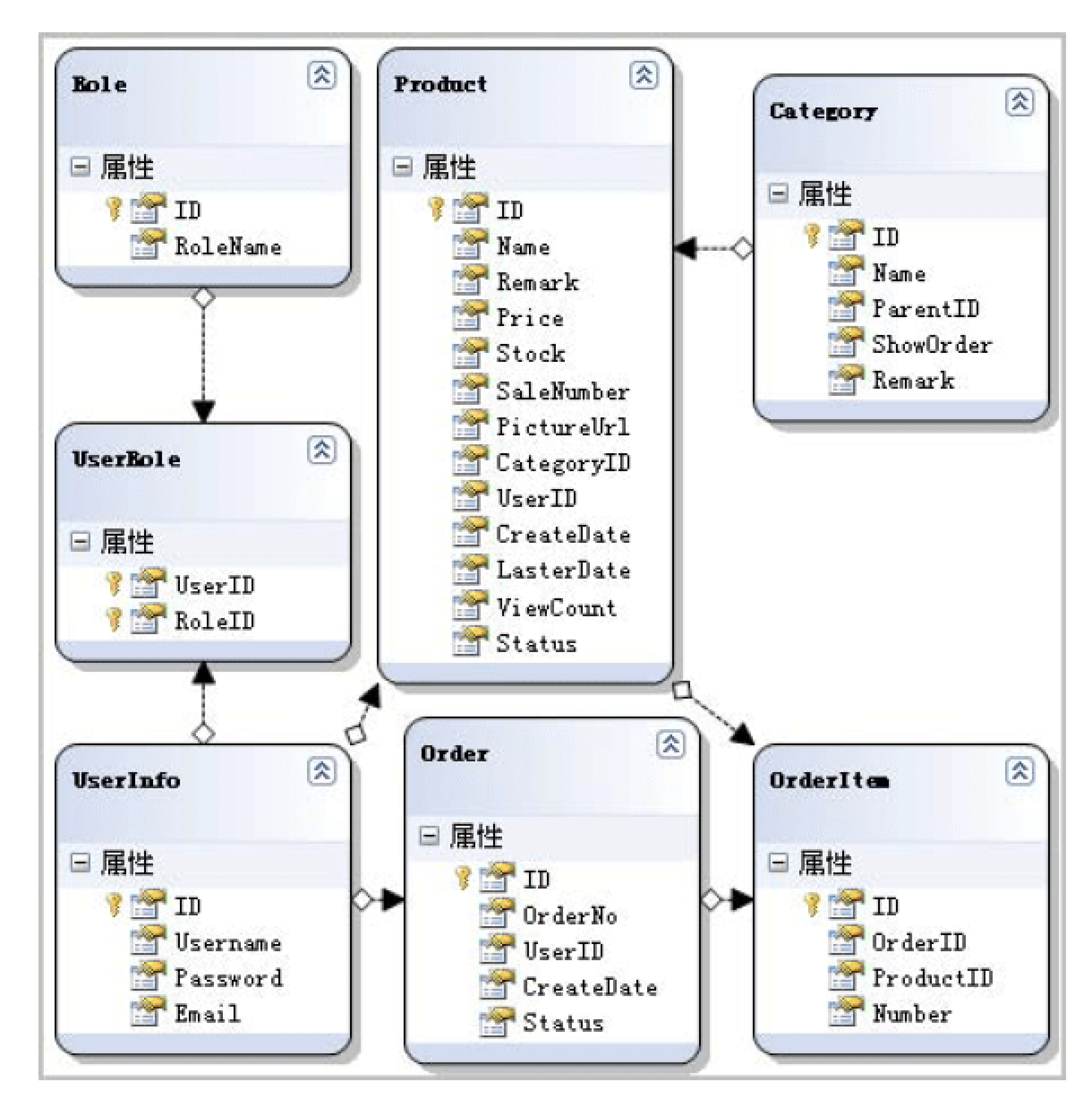


图 1-5

1.3　数据上下文

DataContext 又称为数据上下文，它为 LINQ to SQL 提供操作数据库的入口。如果使用 LINQ to SQL 操作数据库，则首先需要为该数据库创建一个继承于 DataContext 类的自定义的数据上下文类，并在该类中定义表，以及操作数据的方法等。

1.3.1　DataContext 概述

DataContext 类是一个 LINQ to SQL 类，它充当 SQL Server 数据库与映射到该数据库的 LINQ to SQL 实体类之间的管道，它包含用于连接数据库以及操作数据库数据的连接字符串信息和方法。DataContext 类能够通过数据库连接或连接字符串来映射数据库中的所有实体的源，并跟踪和标识用户对数据库的更改。用户可以调用其 SubmitChanges()方法将所有更改提交到数据库。

当 DataContext 类(或从其派生的类)的实例操作数据库时，实例会自动打开和关闭数据库的连接。如果用户显式打开了实例的连接，那么也必须显式关闭实例的连接。

1.3.2　DataContext 类的属性

DataContext 类包括多个属性，如连接属性 Connection、事务属性 Transaction 等。

1. 连接属性 Connection

Connection 属性可以获取 DataContext 类的实例的连接(类型为 DbConnection)。值得注意的是，用户获取该属性的值(即连接对象)之后，该连接对象的默认状态是关闭的。因此，用户如果要使用该连接对象，则需要显式打开该连接对象的状态。

2. 事务属性 Transaction

Transaction 属性为 DataContext 类的实例设置访问数据库的事务。其中，LINQ to SQL 支持以下 3 种事务。



(1) 显式事务。如果 DataContext 类的实例显式设置了 Transaction 属性的值，那么，DataContext 类的实例调用 SubmitChanges()方法时将在同一个事务上进行。

(2) 隐式事务。当 DataContext 类的实例调用 SubmitChanges()方法时，LINQ to SQL 会检查该方法是否在已经指定的事务内，如果不是，则 LINQ to SQL 将启动本地事务，并使用此事务执行所生成的 SQL 命令。

(3) 显式可分发事务。DataContext 类的实例可以在 Transaction 属性指定的事务内调用 LINQ to SQL API，而不会创建新的事务。

3. 执行命令的最大时间属性 CommandTimeout

CommandTimeout 属性可以设置或获取 DataContext 类的实例的查询数据库操作的超时期限。该时间的单位为秒，默认值为 30 秒。有时，查询数据库操作可能需要很长的时间。此时，则需要增大该属性的值，以保证查询数据库的操作能够完成。

1.3.3 DataContext 类的方法

DataContext类包括多个方法，如创建数据库的 CreateDatabase()方法、检测数据库是否存在的 DatabaseExists()方法、删除数据库的 DeleteDatabase()方法、执行 SQL 命令的 ExecuteCommand()方法、执行 SQL 查询的 ExecuteQuery()方法等。

本节将重点讲解 ExecuteCommand()方法与 ExecuteQuery()方法。

1. 执行 SQL 命令的 ExecuteCommand()方法

ExecuteCommand()方法能够执行指定的 SQL 语句，并通过该 SQL 语句来操作数据库。ExecuteCommand()方法返回一个整数值，即该 SQL 语句影响记录的数量。

下面的实例代码调用 ExecuteCommand()方法执行指定的 SQL 语句。具体步骤如下：

(1) 创建 LinqDBDataContext 类的实例 db。

(2) 创建被执行的 SQL 语句。

(3) 调用 ExecuteCommand()方法执行上述指定的 SQL 语句。

(4) 显示 ExecuteCommand()方法执行的结果，即被修改记录的数量。

```
/// <summary>
/// 执行 SQL 命令的 ExecuteCommand()方法
/// </summary>
private void ExecuteCommandWithParam()
{
    //创建 LinqDB 数据库上下文的实例
    LinqDBDataContext db = new LinqDBDataContext(
                                   LinqSystem.LinqDBConnectionString);
    //创建 SQL 语句
    string sql = "UPDATE UserInfo SET Username = {0} WHERE ID > 90";
    //执行 SQL 语句，并返回 SQL 语句影响的行数
    int result = db.ExecuteCommand(sql, "New Name");
    //显示执行的结果
    Response.Write(result.ToString() + "条数据被修改！ ");
}
```

执行结果如图 1-6 所示。



图 1-6

2. 执行 SQL 查询的 ExecuteQuery()方法

ExecuteQuery()方法可以执行指定的 SQL 查询语句，并通过 SQL 查询语句检索数据，查询结果保存数据类型为 IEnumerable 或 IEnumerable<TResult>的对象。

下面的实例代码调用 ExecuteQuery()方法执行指定的 SQL 查询语句。具体步骤如下：



(1) 创建 LinqDBDataContext 类的实例 db。

(2) 创建被执行的 SQL 查询语句。

(3) 调用 ExecuteQuery()方法执行上述指定的 SQL查询语句，查询结果保存在 result 变量中。其中，查询结果中的元素的数据类型为 UserInfo。

(4) 使用 foreach 语句显示 result 变量中的信息。

```
/// <summary>
/// 执行 SQL 查询的 ExecuteQuery()方法
/// </summary>
private void ExecuteSqlQuery()
{
    //创建 LinqDB 数据库上下文的实例
    LinqDBDataContext db = new LinqDBDataContext(
                                  LinqSystem.LinqDBConnectionString);
    //创建 SQL 语句
    string sql = "SELECT TOP 5 * FROM UserInfo";

    IEnumerable<UserInfo> users
        = db.ExecuteQuery<UserInfo>(sql);
    //显示查询的结果
    foreach (UserInfo u in users)
    {
        Response.Write("用户名称： " + u.Username + "<br />");
    }
}
```

执行结果如图 1-7 所示。

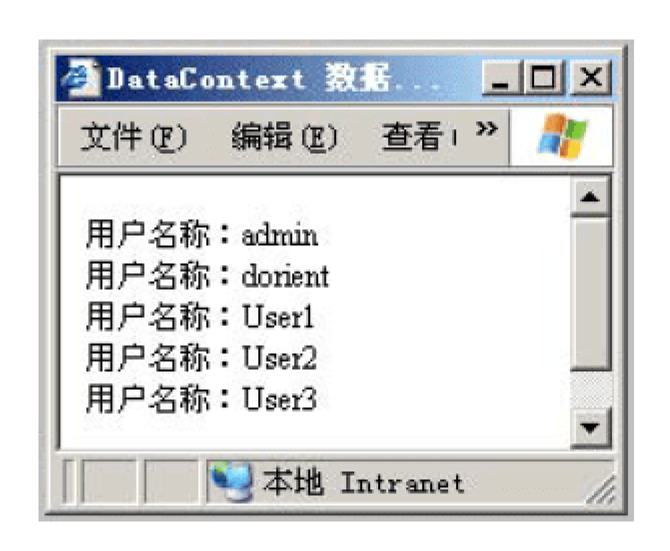


图 1-7

3. 提交更改到数据库的 SubmitChanges()方法

SubmitChanges()方法能够计算要插入、更新或删除的已修改对象的集，并执行相应的修改提交到数据库，并修改数据库。

4. 获取表集合的 GetTable()方法

GetTable()方法能够获取 DataContext 类的实例的表的集合。

5. 获取已修改对象的 GetChangeSet()方法

GetChangeSet()方法能够获取被修改的对象，它返回由 3 个只读集合(Inserts、Deletes 和 Updates)组成的对象。GetChangeSet()方法返回值的类型为 ChangeSet，由 3 个只读集合(Inserts、Deletes 和 Updates)组成。ChangeSet 类包含以下 3 个属性。

- Deletes 属性，已从 ChangeSet 中删除的实体。
- Inserts 属性，已插入到 ChangeSet 中的实体。
- Updates 属性，已在 ChangeSet 中更新的实体。

1.4 处理 Table<T>类型的结果

使用 Table<T>能够方便地操作数据库中的数据，如插入数据到数据库、修改数据库中的数据、删除数据库中的数据，如表 1-2 所示。



表 1-2　Table<T>主要操作数据库的方法

方法名称	说　　明
DeleteAllOnSubmit (TSubEntity)	将集合中的所有实体置于 pending delete 状态
DeleteOnSubmit	将此表中的实体置为 pending delete 状态
InsertAllOnSubmit (TSubEntity)	将集合中所有处于 pending insert 状态的实体添加到 DataContext
InsertOnSubmit	将处于 pending insert 状态的实体添加到此 Table(TEntity)



注意

表中的 4 种方法仅仅只是改变了 Table<T>中 T 的状态，并没有真正地将数据操作的结果写入到数据库中，提交到数据库必须调用相应数据上下文的 SubmitChanges()。

下面的代码，将添加一条数据到 Role 表中。

```
/// <summary>
/// 添加一个新的角色信息到 Role 表中
/// </summary>
private void InsertRole()
{
    //创建 LinqDB 数据库的数据上下文实例
    LinqDBDataContext db = new LinqDBDataContext(
                                LinqSystem.LinqDBConnectionString);
    Response.Write("添加前角色的数量：" + db.Role.Count() + "<br />");
    //创建一个新的角色
    Role role = new Role { RoleName = "新的角色" };
    //将新的角色添加到数据库中
    db.Role.InsertOnSubmit(role);
```

```
    db.SubmitChanges();
    Response.Write("添加后角色的数量：" + db.Role.Count());
}
```

代码运行结果如图 1-8 所示。

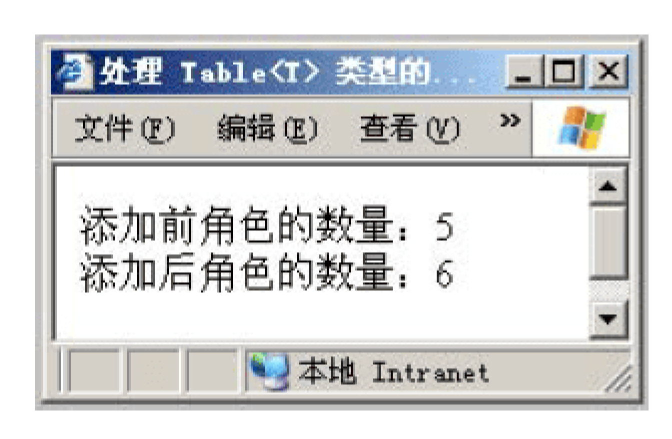


图 1-8

1.5 处理 EntitySet<T>类型的结果

EntitySet<T>用来描述实体类中包含的其他实体的集合。本节介绍了处理 EntitySet<T>类型的结果的方法，以及 EntitySet<T>类的属性和方法。其中，EntitySet<T>类包含 4 个属性和多个方法，如获取实体数量的 Count 属性、获取或设置实体项集合的 Item 属性、添加实体的 Add()方法、移除实体的 Remove()方法等。

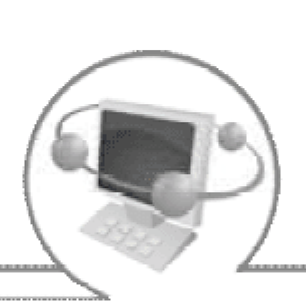
1.5.1 添加实体的 Add()方法

Add()方法能够将元素或实体添加到 EntitySet<T>集合中。新添加的元素或实体位于 EntitySet<T>集合的末尾处。

下面的实例代码演示了添加新实体到 productsforFirstUser 实例(数据类型为 EntitySet<Product>)中的方法，并显示了添加新实体之前和之后 productsforFirstUser 实例中的实体的数量。具体步骤如下：

(1) 创建 LinqDBDataContext 类的实例 db。

(2) 获取与第 1 个用户相关的商品信息，并保存为 productsforFirstUser 实例。其中，该实例的数据类型为 EntitySet<Product>。



(3) 显示添加新的实体之前 productsforFirstUser 实例中的实体的数量。

(4) 创建一个新的实体 p(实体的数据类型为 Product)，并设置了该实例的属性的值。

(5) 调用 Add()方法把实体 p 添加到 productsforFirstUser 实例中。

(6) 调用 SubmitChanges()方法将上述修改提交到数据库中。

(7) 显示添加新的实体之后 productsforFirstUser 实例中的实体的数量。

```
/// <summary>
/// 添加实体的 Add()方法
/// </summary>
private void ShowEntitySetAdd()
{
    //创建 LinqDB 数据库上下文的实例
    LinqDBDataContext db = new LinqDBDataContext(
                                      LinqSystem.LinqDBConnectionString);
    //获取第一个用户的商品信息
    EntitySet<Product> productsforFirstUser
        = db.UserInfo.First<UserInfo>().Product;
    //显示商品数量
    Response.Write("添加商品之前的数量:"
        + productsforFirstUser.Count.ToString() + "<br />");
    //创建一个新对象
    Product p = new Product();
    p.Name = "New Product" + DateTime.Now.ToString();
    p.Price = 100.0m;
    p.LasterDate = DateTime.Now;
    p.PictureUrl = string.Empty;
    p.Remark = string.Empty;
```

```
    p.SaleNumber = 0;
    p.Status = 1;
    p.Stock = 100;
    p.UserID = 1;
    p.ViewCount = 0;
    p.CategoryID = 2;
    p.CreateDate = DateTime.Now;
    //添加对象到 productsforFirstUser 集合中
    productsforFirstUser.Add(p);
    //提交更改到数据库
    db.SubmitChanges();
    //显示商品数量
    Response.Write("添加商品之后的数量:" +
        productsforFirstUser.Count.ToString() + "<br />");
}
```

运行结果如图 1-9 所示。

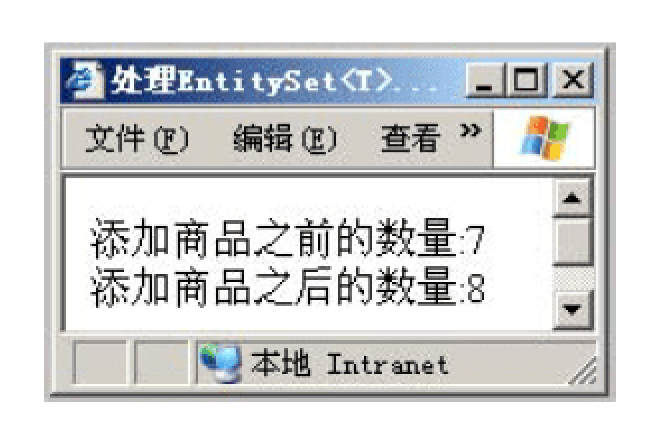


图 1-9

1.5.2 移除实体的方法

Remove()方法能够从 EntitySet<T>集合中移除指定的元素或实体。如果移除成功则返回 true；否则返回 false。

RemoveAt()方法能够从 EntitySet<T>集合中移除指定位置处的元素或实体。



下面的实例代码演示了从 productsforFirstUser 实例(数据类型为 EntitySet<Product>)中移除最后一个元素或实体的方法，并显示了移除实体之前和之后 productsforFirstUser 实例中的实体的数量。具体步骤如下：

(1) 创建 LinqDBDataContext 类的实例 db。

(2) 获取与第 1 个用户相关的商品信息，并保存为 productsforFirstUser 实例。其中，该实例的数据类型为 EntitySet<Product>。

(3) 显示移除实体之前 productsforFirstUser 实例中的实体的数量。

(4) 调用 RemoveAt()方法从 productsforFirstUser 实例中移除最后一个实体。

(5) 调用 SubmitChanges()方法将上述修改提交到数据库中。

(6) 显示移除实体之后 productsforFirstUser 实例中的实体的数量。

```
/// <summary>
/// 移除指定索引的实体 RemoveAt()方法
/// </summary>
private void ShowEntitySetRemoveAt()
{
    //创建 LinqDB 数据库上下文的实例
    LinqDBDataContext db = new LinqDBDataContext(
                                LinqSystem.LinqDBConnectionString);
    //获取第一个用户的商品信息
    EntitySet<Product> productsforFirstUser
        = db.UserInfo.First<UserInfo>().Product;
    //显示商品数量
    Response.Write("移除商品之前的数量:"
        + productsforFirstUser.Count.ToString() + "<br />");
    //使用 RemoveAt()移除集合中的最后一个对象
    productsforFirstUser.RemoveAt(productsforFirstUser.Count - 1);
```

```
    //提交更改到数据库
    db.SubmitChanges();
    //显示商品数量
    Response.Write("移除商品之后的数量:" +
        productsforFirstUser.Count.ToString() + "<br />");
}
```

运行结果如图 1-10 所示。

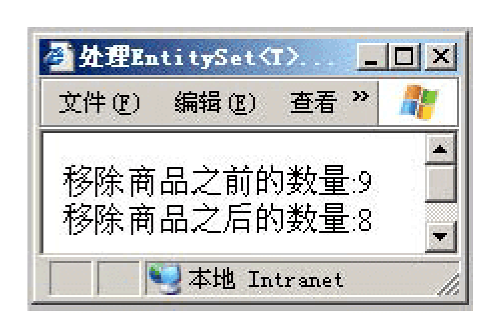


图 1-10

1.5.3　查找是否包含实体的 Contains()方法

Contains()方法能够判断在 EntitySet<T>集合中是否包含指定的元素或实体。如果 EntitySet<T>集合包含指定的元素或实体，则返回 true；否则返回 false。

1.6　处理 EntityRef<T>类型的结果

EntityRef<T>类用来处理一对多关系中的数据。不妨考虑 LinqDB 数据库，在该数据库中，Order 表的 UserID 字段引用 UserInfo 表的 ID 字段作为外键。如果要获取 Order 表中某一个记录相关的用户信息，则可以使用 EntityRef<T>类来处理。LinqDBDataContext 类(LinqDB 数据库的数据上下文类)中的 Order 类(与 Order 表相对应)中的 UserInfo 属性的类型为 EntityRef<UserInfo>。UserInfo 属性描述了 Order 表中某一个记录的用户信息。定义 UserInfo 属性(属于 Order 类)的程序代码如下：



```
[Table(Name="dbo.[Order]")]
public partial class Order : INotifyPropertyChanging, INotifyPropertyChanged
{
    private EntityRef<UserInfo> _UserInfo;
    [Association(Name="UserInfo_Order", Storage="_UserInfo",
            ThisKey="UserID", IsForeignKey=true)]
  public UserInfo UserInfo
  {
        get    {return this._UserInfo.Entity;}
        set    {...}
    }
}
```

当然，在使用的时候，已经由编译器为我们自动生成了上述的代码。不需要由程序员去手动编写这段代码。

下面的实例代码显示了 Order 表中所有记录相关的用户信息，即 Order 表中所有记录与 UserInfo 相关的关系数据。具体步骤如下：

(1) 创建 LinqDBDataContext 类的实例 db。

(2) 使用 foreach 语句显示 Order 表中所有订单的信息(如订单 ID 和订单编号)。

(3) 获取 Order 表中每一个订单相关的用户信息(UserInfo 属性的值，由 EntityRef<T> 类定义)，并显示相关用户的信息(如用户 ID、用户名称和 E-mail)。

```
/// <summary>
/// 处理 EntityRef<T>类型的结果
/// </summary>
private void ShowEntityRefInfo()
{
```

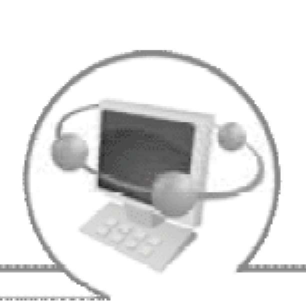
```
    //创建 LinqDB 数据库上下文的实例
    LinqDBDataContext db = new LinqDBDataContext(
                            LinqSystem.LinqDBConnectionString);
    //获取所有的订单信息
    List<Order> orders = db.Order.ToList();
    //显示订单信息
    foreach (Order order in orders)
    {
        Response.Write("Order ID:" + order.ID.ToString() + ",");
        Response.Write("Order No:" + order.OrderNo.ToString()
                            + "<br />");
        //获取该订单的用户信息
        UserInfo ui = order.UserInfo;
        Response.Write("User ID:" + ui.ID.ToString() + ",");
        Response.Write("Username:" + ui.Username + ",");
        Response.Write("E-mail:" + ui.Email + "<br /><hr />");
    }
}
```

运行结果如图 1-11 所示。



图 1-11

另外，EntityRef<T>类包含 2 个属性：Entity 和 HasLoadedOrAssignedValue。其中，Entity 属性可以获取 EntityRef<T>类的实例的实体，HasLoadedOrAssignedValue 属性表示 EntityRef<T>类的实例是否加载或分配关系数据。



1.7　处理 ISingleResult<T>类型的结果

ISingleResult<T>泛型接口表示具有单个返回序列的映射函数的结果。在 SQL 数据实体类中，该类常常使用 ISingleResult<T>泛型接口来保存存储过程(或函数等)检索数据库的结果。特别地，当存储过程(或函数等)返回多个检索结果时。

下面的实例代码创建了一个名称为 Pr_GetCategorys 的存储过程。该存储过程从 Category 表中检索一个商品分类的 ID 值、名称、父分类的 ID 值、显示顺序、父分类的名称、子分类的数量、兄弟分类的数量。检索结果按照显示顺序(ShowOrder)进行排序。

```
CREATE PROC Pr_GetCategorys
AS SELECT A.ID,A.Name,A.ParentID,A.ShowOrder,A.Remark,
   ISNULL((SELECT Name FROM Category AS B WHERE A.ParentID = B.ID),null) AS ParentName,
   (SELECT COUNT(*) FROM Category AS C WHERE A.ID = c.ParentID) AS SubCount,
   (SELECT COUNT(*) FROM Category AS D WHERE A.ParentID = D.ParentID) AS SiblingCount
FROM Category AS A ORDER BY ParentID,ShowOrder
```

下面的代码调用上一节介绍的 Pr_GetCategorys()方法从 Category 表中获取商品分类的数据，并显示商品分类的 ID 值、名称、父分类的 ID 值、显示顺序、父分类的名称、子分类的数量、兄弟分类的数量等数据。

```
/// <summary>
/// 从 Category 表中获取商品分类的数据
/// </summary>
private void ShowCategoryInfo()
{
     //创建 LinqDB 数据库上下文的实例
     LinqDBDataContext db = new LinqDBDataContext
```

```
                    (LinqSystem.LinqDBConnectionString);
    //获取存储过程返回的查询结果
    ISingleResult <Pr_GetCategorysResult> result
              = db.Pr_GetCategorys();
    //显示查询的信息
    foreach (Pr_GetCategorysResult r in result)
    {
        Response.Write("ID:" + r.ID.ToString() + "，");
        Response.Write("Name:" + r.Name + "，");
        Response.Write("ParentID:" + r.ParentID.ToString() + "，");
        Response.Write("ParentName:" + r.ParentName + "，");
        Response.Write("ShowOrder:" + r.ShowOrder.ToString() + "，");
        Response.Write("SiblingCount:"
                   + r.SiblingCount.ToString() + "，");
        Response.Write("SubCount:" + r.SubCount.ToString() + "，");
        Response.Write("Remark:" + r.Remark + "。<br /><hr />");
    }
}
```

运行结果如图 1-12 所示。

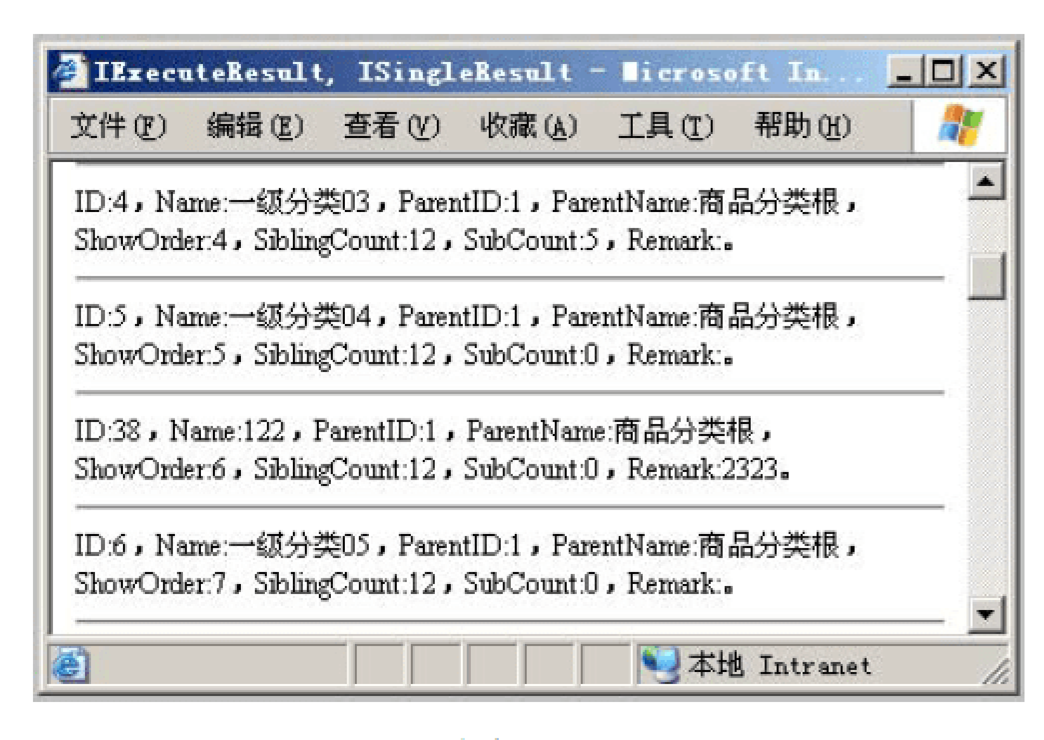


图 1-12



1.8　查询数据库中的数据

使用 LINQ to SQL 查询 SQL Server 数据库中的数据，一般要为该数据库创建一个 DBML 文件，并为该数据库创建数据上下文类。

1.8.1　简单查询

使用 LINQ to SQL 可以轻松查询数据库中的数据，比传统的 SQL 语句或存储过程查询数据库的方法更加简洁。下面的实例代码使用 LINQ to SQL 查询 LinqDB 数据库的 UserInfo 表中的数据，具体步骤如下。

(1) 创建 LinqDBDataContext 类的实例 db。

(2) 使用 LINQ 查询表达式查询 UserInfo 表中 ID 列的值小于 10，且 Username 列的值的长度大于 5 的数据。查询结果保存为 result 变量。

(3) 把 result 变量设置为 GridView 控件的数据源，并绑定该控件的数据，显示查询结果。

```
/// <summary>
/// 简单查询
/// </summary>
private void ShowData()
{
    //创建 LinqDB 数据库的数据上下文实例
    LinqDBDataContext db = new LinqDBDataContext(
                            LinqSystem.LinqDBConnectionString);
    //使用 LINQ 查询表达式查询数据
    var result = from user in db.UserInfo
                 where user.ID < 10 && user.Username.Length > 5
```

```
                    select user;
    //设置数据源，并绑定数据
    gvUserInfo.DataSource = result;
    gvUserInfo.DataBind();
}
```

运行结果如图 1-13 所示。

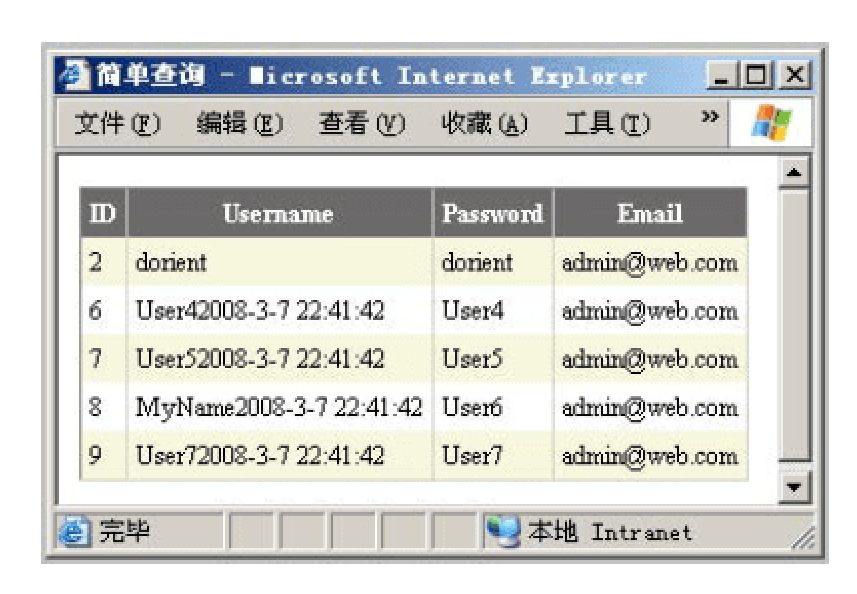


图 1-13

1.8.2　复杂查询

上一小节使用简单的 LINQ 查询表达式查询 LinqDB 数据库的 UserInfo 中的数据，且该查询中只涉及一个表(UserInfo)，查询条件也相对简单。本小节介绍比较复杂的 LINQ 查询表达式，并在 LINQ 查询表达式中使用 join 子句联接多个相关的表。

下面的实例代码使用 LINQ to SQL 查询 LinqDB 数据库中的 UserInfo、UserRole 和 Role 表中的数据，并在查询中使用 join 子句联接相关的表。具体步骤如下。

(1) 创建 LinqDBDataContext 类的实例 db。

(2) 使用 LINQ 查询表达式查询 UserInfo 表中 ID 列的值小于 10，且 Username 列的值的长度大于 5 的数据。

(3) 将上述 LINQ 查询表达式的查询结果保存为 result 变量。其中，结果由 UserInfo 表的 Username 列的值和 Role 表的 Rolename 列的值组成。

(4) 把 result 变量设置为 GridView 控件的数据源，并绑定该控件的数据，显示查询结果。



```
/// <summary>
/// 复杂查询
/// </summary>
private void ShowData()
{
    //创建 LinqDB 数据库的数据上下文实例
    LinqDBDataContext db = new LinqDBDataContext(
                                        LinqSystem.LinqDBConnectionString);
    //使用 LINQ 查询表达式查询数据
    var result = from user in db.UserInfo
                    join ur in db.UserRole on user.ID equals ur.UserID
                    join role in db.Role on ur.RoleID equals role.ID
                    where user.ID < 10 && user.Username.Length > 5
                    select new { user.Username, role.RoleName };
    //设置数据源，并绑定数据
    gv.DataSource = result;
    gv.DataBind();
}
```

运行结果如图 1-14 所示。



图 1-14

1.8.3　聚合查询

使用 LINQ to SQL 不但可以轻松查询数据库中的数据，而且还可以对查询结果

进行聚合计算，如计算查询结果的和(SUM)、最大值(Max)、最小值(Min)、平均值(Average)等。

下面的实例代码使用 LINQ to SQL 查询 LinqDB 数据库的 Product 表中的数据，并计算查询结果中商品的最高价格和最低价格。具体步骤如下。

(1) 创建 LinqDBDataContext 类的实例 db。

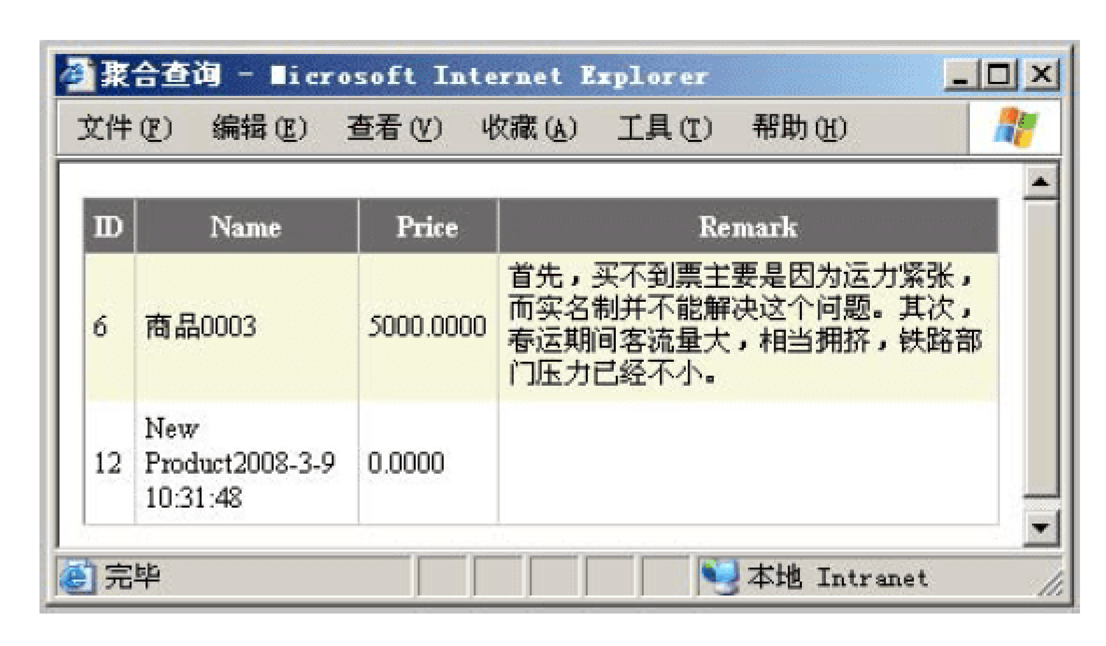
(2) 使用 LINQ 查询表达式查询 Product 表中的数据(商品)，同时，分别调用 Max 和 Min 操作计算商品的最高价格和最低价格，查询结果保存为 result 变量。

(3) 把 result 变量设置为 GridView 控件的数据源，并绑定该控件的数据，显示查询结果。

```
/// <summary>
/// 聚合查询
/// </summary>
private void ShowData()
{
    //创建 LinqDB 数据库的数据上下文实例
    LinqDBDataContext db = new LinqDBDataContext(
                        LinqSystem.LinqDBConnectionString);
    //使用 LINQ 查询表达式查询数据
    var result = from p in db.Product
                 where p.Price == db.Product.Max(c => c.Price.Value)
                 || p.Price == db.Product.Min(c => c.Price.Value)
                 select new { p.ID, p.Name, p.Price, p.Remark };
    //显示最高和最低价格的商品
    gv.DataSource = result;
    gv.DataBind();
}
```



运行结果如图 1-15 所示。



ID	Name	Price	Remark
6	商品0003	5000.0000	首先，买不到票主要是因为运力紧张，而实名制并不能解决这个问题。其次，春运期间客流量大，相当拥挤，铁路部门压力已经不小。
12	New Product2008-3-9 10:31:48	0.0000	

图 1-15

1.8.4　分组查询

分组查询和 SQL 语句中的带有“group”关键字的查询功能类似，它能够把查询结果按照关键字进行分组。下面的实例代码使用 LINQ to SQL 查询 LinqDB 数据库的 Product 表中的数据，并按照 UserID 列的值对查询结果进行分组和排序。具体步骤如下。

(1) 创建 LinqDBDataContext 类的实例 db。

(2) 使用 LINQ 查询表达式查询 Product 与 UserInfo 表并联接起来，然后对联接起来的子查询数据按照 UserID 进行分组，并将查询结果保存为 result 变量。

(3) result 是一个已经分组的数据源，因此，不能将 result 与 GridView 数据控件直接绑定。必须将 result 中的每一组取出来分别绑定到一个 GridView 数据源控件，显示查询结果。

```
/// <summary>
/// 分组查询
/// </summary>
private void ShowData()
{
    //创建 LinqDB 数据库的数据上下文实例
    LinqDBDataContext db = new LinqDBDataContext(
                    LinqSystem.LinqDBConnectionString);
```

```
    //使用 LINQ 查询表达式查询数据
    var result = from pu in
             (from u in db.UserInfo
              join p in db.Product on u.ID equals p.UserID
              select new { p.ID, p.Name, p.Price, p.UserID, u.Username })
                      group pu by pu.UserID;
    //设置数据源，并绑定数据
    foreach (var r in result)
    {
        GridView gv = new GridView();
        gv.ID = "gv" + r.Key.ToString();
        gv.DataSource = r ;
        gv.DataBind();
        //Pnl 为 Panel 控件
        Pnl.Controls.Add(gv);
        Pnl.Controls.Add(new LiteralControl("<hr />"));
    }
}
```

运行上面的代码，结果如图 1-16 所示。

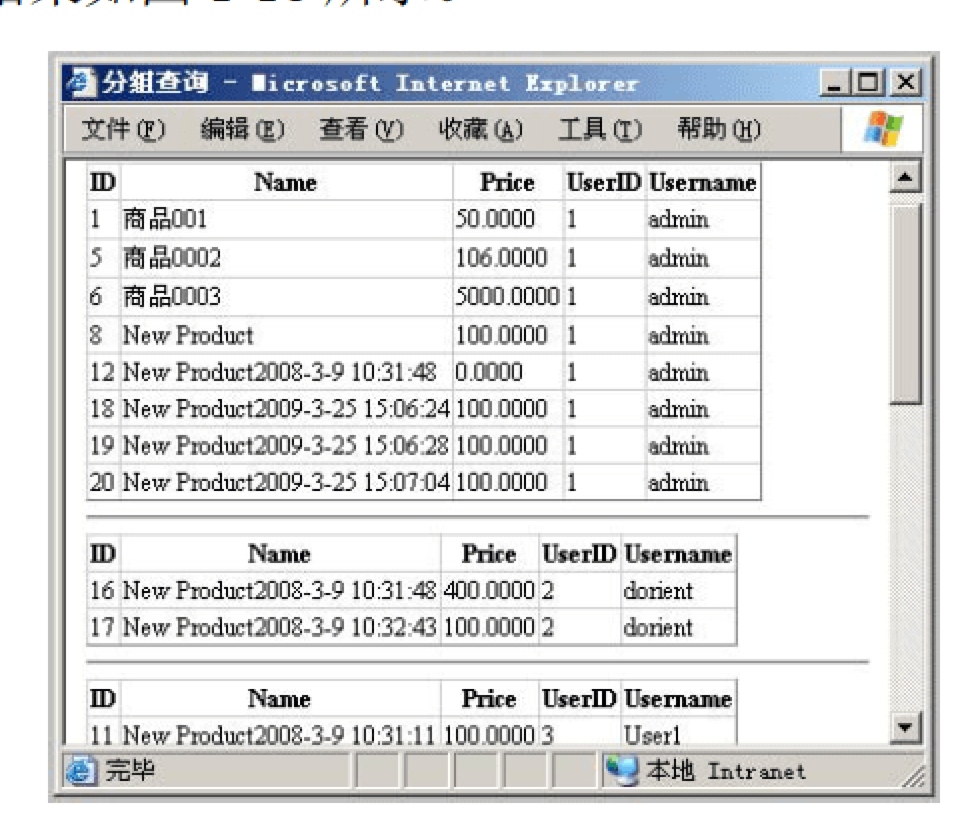


图 1-16



1.9　动态数据支持

ASP.NET 3.5 Extensions CTP 包含了一个新特性是 ASP.NET Dynamic Data Support(动态数据支持)，它允许开发人员不用编写一行代码就能极其快速地建造使用 LINQ to SQL 对象模型的数据驱动网站。

1. 创建 ASP.NET Dynamic Data 站点

安装完 ASP.NET 3.5 Extensions 后，可以在新建项目对话框中看到一项 Dynamic Data 网站，如图 1-17 所示。

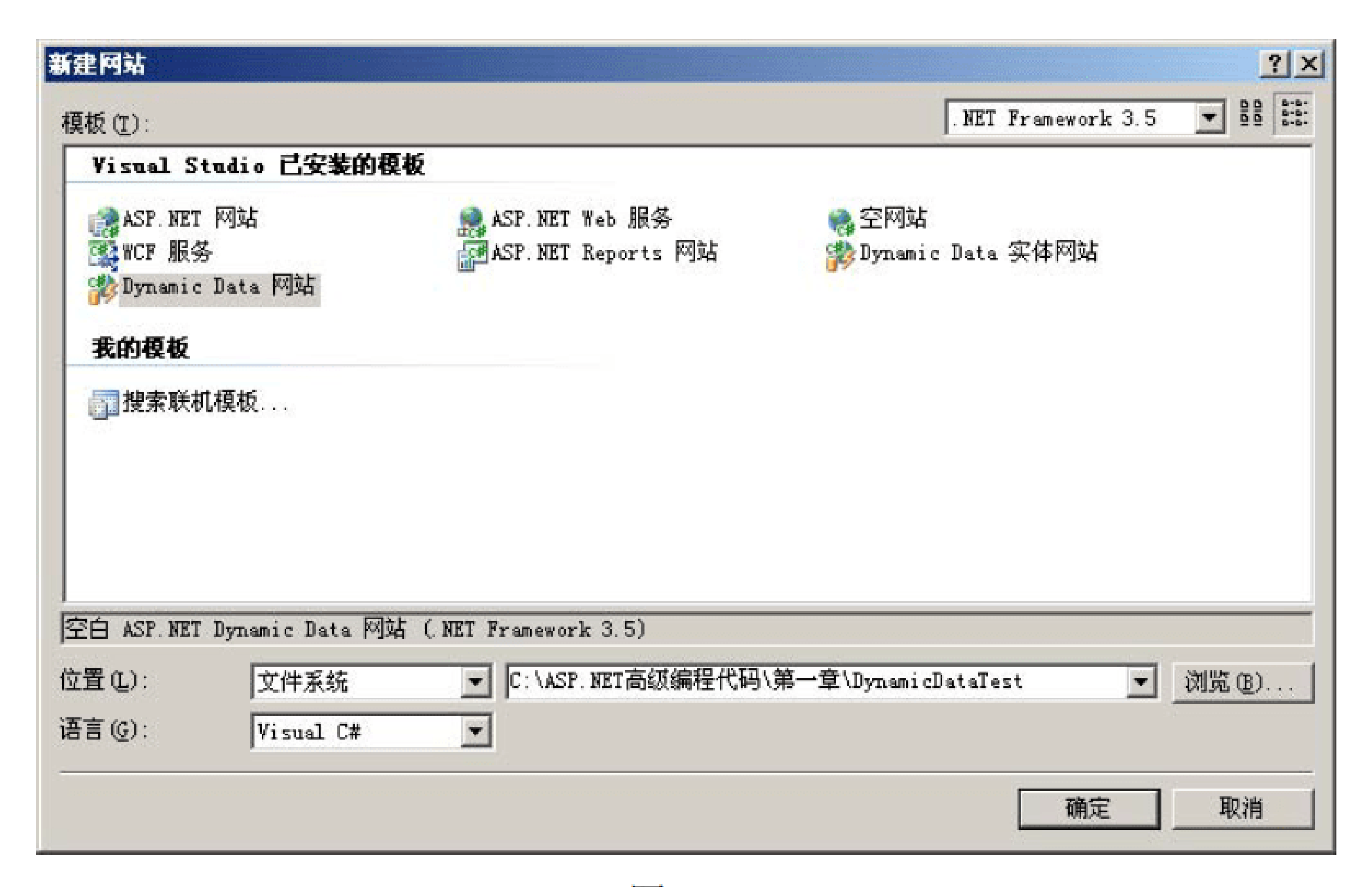


图 1-17

创建完成后，项目结构如下：

在解决方案中会有一个 Dynamic Data 文件夹，该文件夹里存放的都是模板文件，可以通过这些模板文件来定制网站的外观。

2. 添加数据模型

添加一个 LINQ to SQL 文件，以 pubs 的数据库为例，数据上下文类的名称为 DataClasses DataContext，如图 1-18 所示。

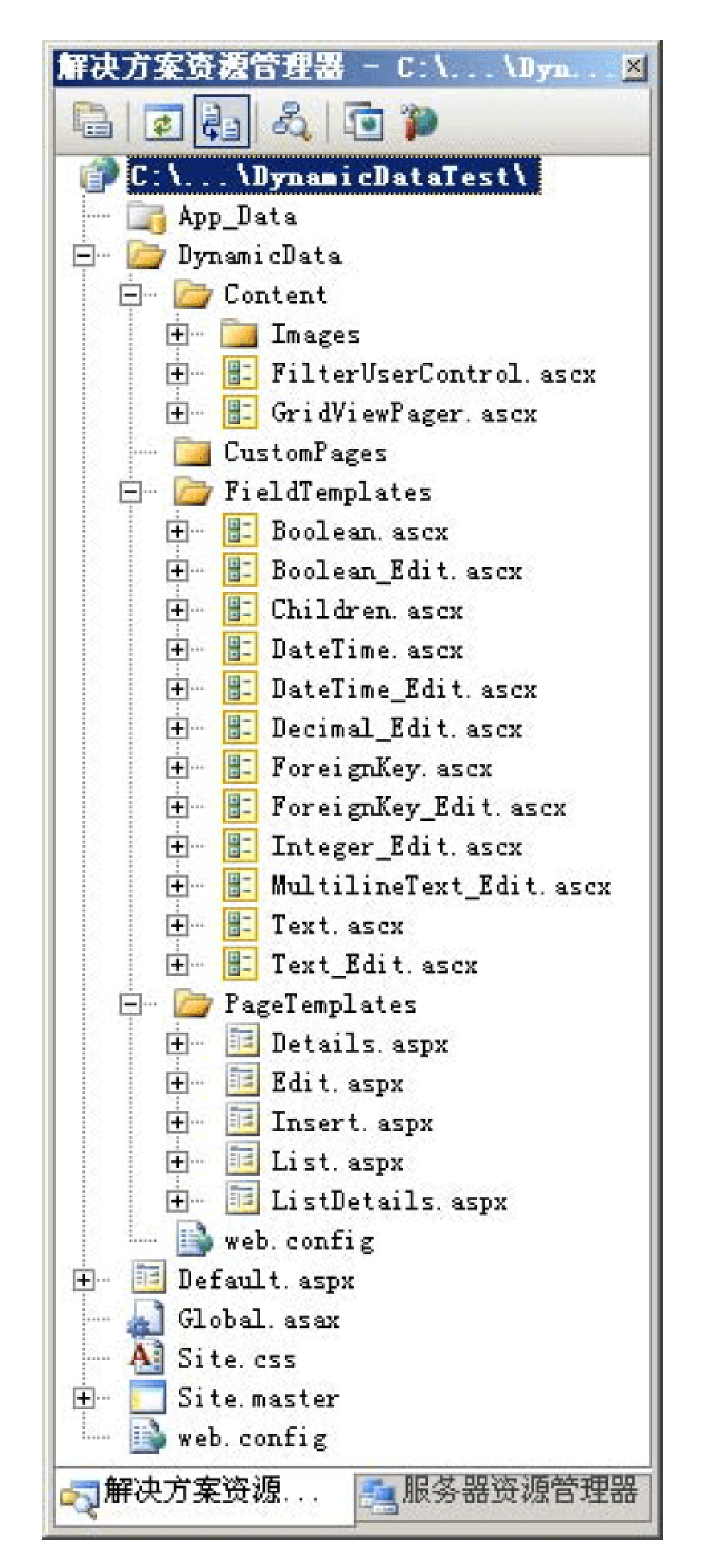


图 1-18

3. 修改 Global.asax 文件

只需要修改一行代码即可，就是注册 LINQ to SQL 生成的动态数据模型，图 1-19 给出了 DataClassesDataContext 数据上下文的表。

```
model.RegisterContext(typeof(DataClassesDataContext),
        new ContextConfiguration() { ScaffoldAllTables = true });
```



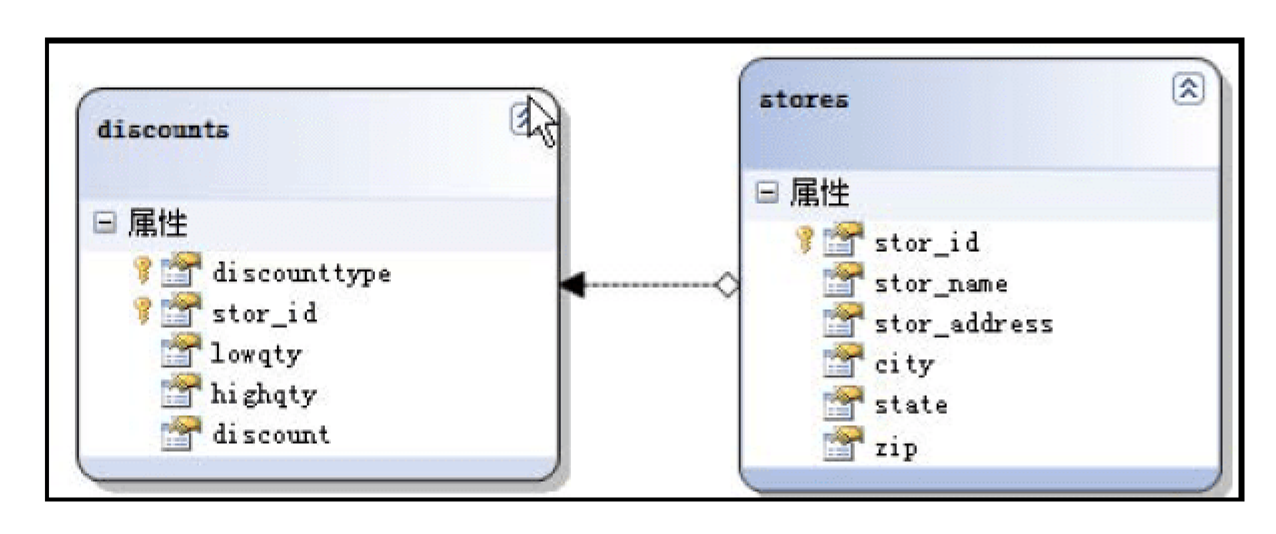


图 1-19

4. 运行站点

(1) 运行站点后，在默认的主页上列出了在数据模型中添加的所有数据表，如图 1-20 所示。

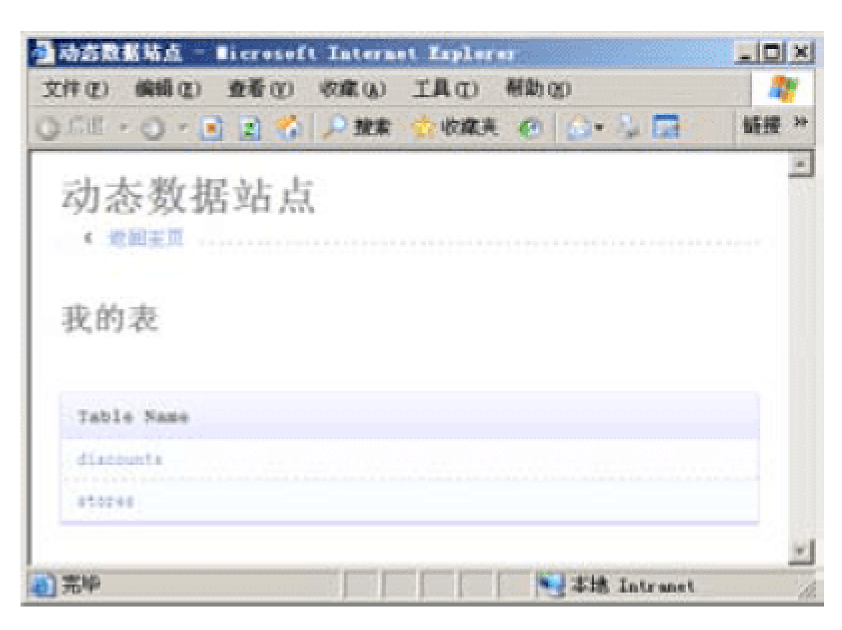


图 1-20

(2) 单击 stores 进入，生成了一个列表界面，显示出 stores 中的数据，并且自动根据外键关系，显示了 discounts 一列，可以导航到 discounts 表的信息，如图 1-21 所示。

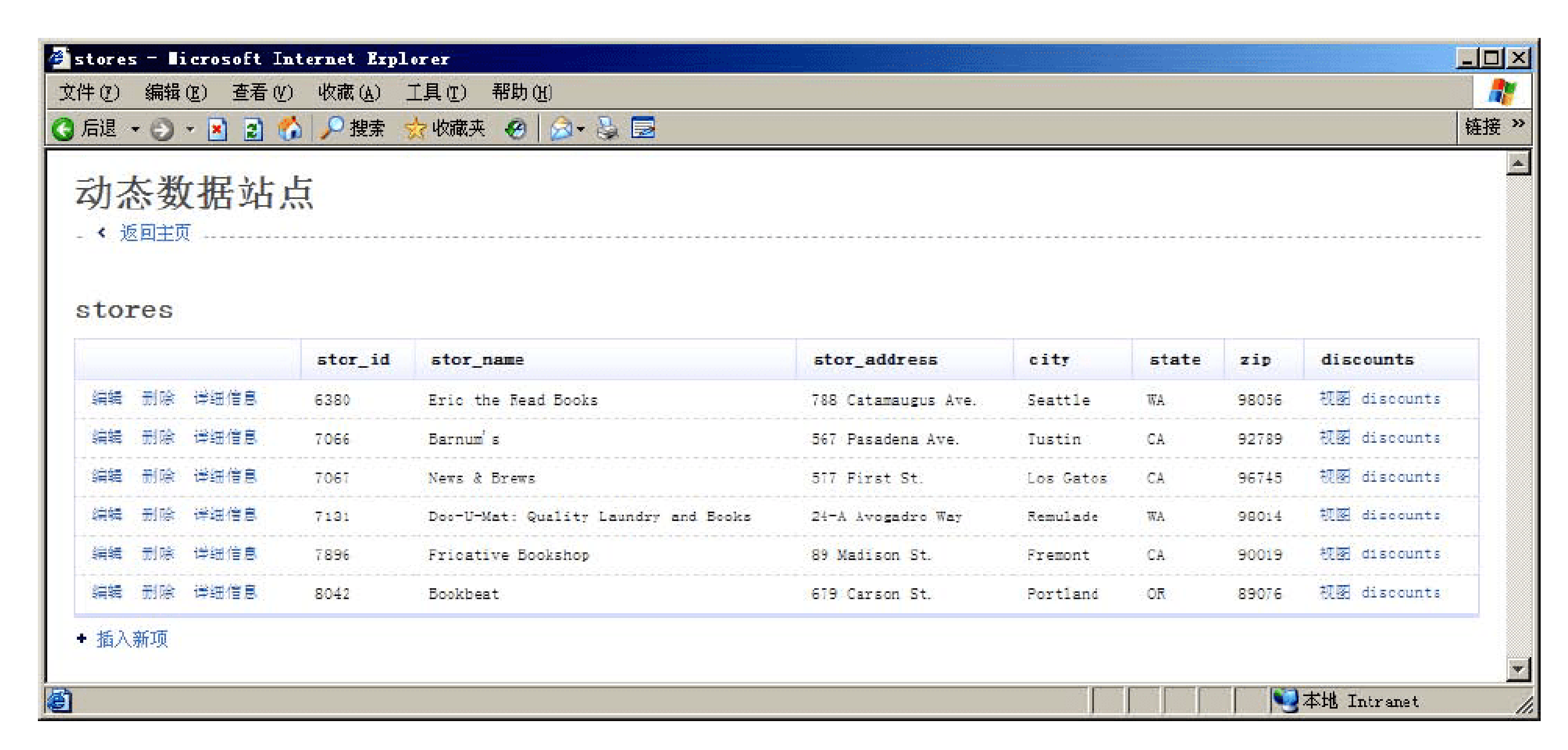


图 1-21

(3) 根据主外键自动列出筛选条件，如图 1-22 所示。

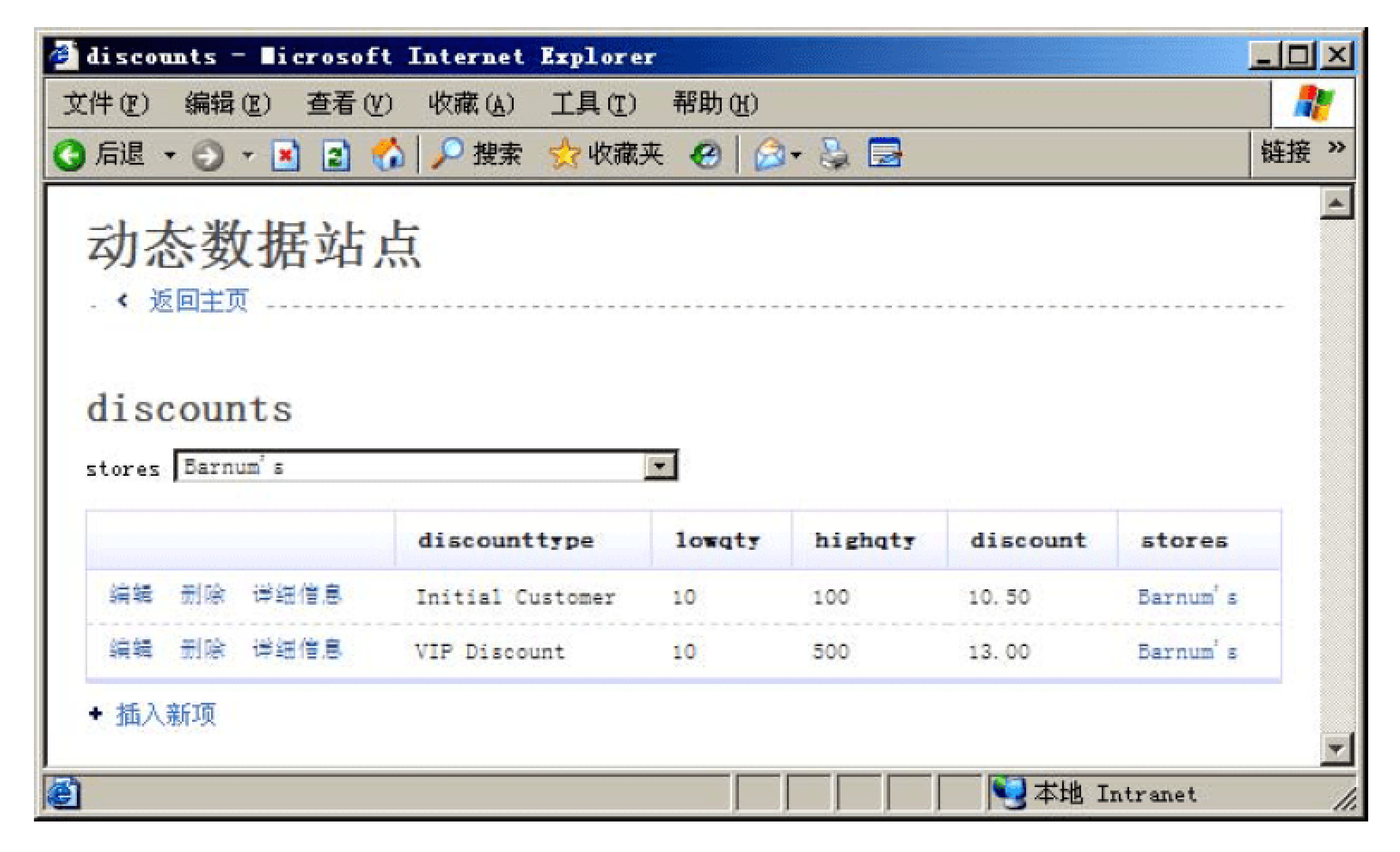


图 1-22

(4) 内置的数据验证支持，如图 1-23 所示。



图 1-23

动态数据支持还可以定制网站，比如修改母版页和 CSS 文件、自定义动态数据视图、自定义动态数据字段和自定义验证等。



【小结】

- DataContext 类能够通过数据库连接或连接字符串来映射数据库中的所有实体的源，并跟踪和标识用户对数据库的更改。
- EntitySet<T>类用来描述实体类中包含的其他实体集合。
- EntityRef<T>类用来处理一对多关系中的数据。
- ISingleResult<T>泛型接口表示具有单个返回序列的映射函数的结果。
- 使用 LINQ 对数据库数据进行复杂查询。
- 使用 LINQ 对数据库数据进行聚合查询。
- 使用 LINQ 对数据库数据进行分组查询。
- 动态数据支持使用 LINQ to SQL 对象模型。

【自测题】

1. (　　)命名空间包含于创建 LINQ to SQL 的所有类。

   A. System.Web.UI.Linq　　B. System.Xml.Linq

   C. System.Web　　D. System.Web.Linq

2. 获取 Table<T>类型的结果的行数可以使用(　　)。

   A. Count 属性　　B. Count 方法

   C. Rows.Count 属性　　D. Rows.Count 方法

3. 通过 LINQ to SQL 修改数据上下文后需要调用数据上下文的(　　)方法才能将修改保存到数据库。

   A. SubmitChanges　　B. GetChangeSet

   C. Update　　D. Save

4. 下列最适合描述“某班级对象里的学生集合”的是(　　)。

A. EntitySet<T>　　B. EntityRef<T>

C. ISingleResult<T>　　D. IMultileResult<T>

5. 关于动态数据支持，下列说法错误的是(　　)。

A. 需要安装 ASP.NET 3.5 Extensions

B. 可以在 ASP.NET 2.0 上运行

C. 可以自定义母版

D. 需要 LINQ to SQL 或 ADO.NET ENTITY

【上机部分】

上机目标

- Linq 在三层结构中的应用
- 使用 LINQ to SQL 操作数据库

上机练习

◆ 第一阶段 ◆

练习：使用 LINQ to SQL 构建一个三层结构应用程序，用来显示和删除新闻信息。

【问题描述】

新闻具有不同的类别，要求可以显示、编辑、删除和添加新闻。

【问题分析】

可以在 Model 中拖放一个 LINQ to SQL 文件来充当实体类，因为 LINQ to SQL 会自动



生产实体类和操作实体的方法。

【参考步骤】

(1) 建立数据库和表，并加入测试数据。

```
create database NewsDB
go
use NewsDB
go
--新闻类别表
create table Type
(
   T_ID int primary key identity(1,1),
   T_Name varchar(20)
)
insert into Type values('娱乐')
insert into Type values('财经')
insert into Type values('体育')
insert into Type values('科技')
--新闻信息表
create table News
(
   N_ID int primary key identity(1,1),
   N_Title varchar(200),
   N_Author varchar(20),
   N_Content nvarchar(2000),
   T_ID int references Type(T_ID)
)
```

(2) 建立解决方案 News，在该解决方案下添加 4 个项目：Model、DAL、BLL 和 Web。添加项目之间的引用关系。

(3) 在 Model 下拖放一个 LINQ to SQL 文件，关联到 NewsDB 数据库。

(4) 在 DAL 项目下添加 TypeService.cs 文件，代码如下：

```
public class TypeService
{
    /// <summary>
    /// 得到所有的类别列表
    /// </summary>
    public static List<Model.Type> GetAllType()
    {
        Model.NewsDBDataContext db = new Model.NewsDBDataContext();
        return db.Type.ToList();
    }
}
```

(5) 在 DAL 项目下添加 NewsService.cs 文件，代码如下：

```
public class NewsService
{
    /// <summary>
    /// 实现添加新闻
    /// </summary>
    public static bool InsertNews(Model.News n)
    {
        try
        {
```



```
            Model.NewsDBDataContext db =
                new Model.NewsDBDataContext();
            db.News.InsertOnSubmit(n);
            db.SubmitChanges();
            return true;
        }
        catch (Exception)
        {
            return false;
        }
    }
    /// <summary>
    /// 实现删除新闻
    /// </summary>
    public static bool DeleteNews(Model.News n)
    {
        try
        {
            Model.NewsDBDataContext db =
                new Model.NewsDBDataContext();
            var newsQuery =
                (from news in db.News
                 where news.N_ID == n.N_ID
                 select news).First();
            db.News.DeleteOnSubmit(newsQuery);
            db.SubmitChanges();
```

```
                return true;
            }
            catch (Exception)
            {
                return false;
            }
        }
        /// <summary>
        /// 实现修改新闻
        /// </summary>
        public static bool UpdateNews(Model.News n)
        {
            try
            {
                Model.NewsDBDataContext db =
                        new Model.NewsDBDataContext();
                var newsQuery =
                    (from news in db.News
                     where news.N_ID == n.N_ID
                     select news).First();
                newsQuery.N_Author = n.N_Author;
                newsQuery.N_Content = n.N_Content;
                newsQuery.N_Title = n.N_Title;
                newsQuery.T_ID = n.T_ID;
                db.SubmitChanges();
                return true;
```



```
        }
        catch (Exception)
        {
            return false;
        }

    }
    /// <summary>
    /// 实现得到所有新闻列表
    /// </summary>
    public static List<Model.News> GetAllNews()
    {
        Model.NewsDBDataContext db = new Model.NewsDBDataContext();
        return db.News.ToList();
    }
    /// <summary>
    /// 根据新闻编号得到单个新闻对象
    /// </summary>
    public static Model.News GetNewsByN_ID(int n_id)
    {
        Model.NewsDBDataContext db = new Model.NewsDBDataContext();
        var newsQuery =
            (from news in db.News
             where news.N_ID == n_id
             select news).First();
        return newsQuery;
```

```
    }
}
```

(6) 在 BLLL 项目下添加 Type.cs 文件，代码如下：

```
public class Type
{
    public static List<Model.Type> GetAllType()
    {
        return DAL.TypeService.GetAllType();
    }
}
```

(7) 在 BLL 项目下添加 News.cs 文件，代码如下：

```
public class News
{
    public static bool InsertNews(Model.News n)
    {
        return DAL.NewsService.InsertNews(n);
    }
    public static bool DeleteNews(Model.News n)
    {
        return DAL.NewsService.DeleteNews(n);
    }
    public static bool UpdateNews(Model.News n)
    {
        return DAL.NewsService.UpdateNews(n);
```



```
    }
    public static List<Model.News> GetAllNews()
    {
        return DAL.NewsService.GetAllNews();
    }
    public static Model.News GetNewsByN_ID(int n_id)
    {
        return DAL.NewsService.GetNewsByN_ID(n_id);
    }
    /// <summary>
    /// 查询某一类型的所有新闻
    /// </summary>
public static System.Collections.Generic.IEnumerable<Model.News>
            GetNewsByT_ID(int t_id)
    {
        List<Model.News> list = GetAllNews();
        if (t_id == -1)
            return list;
        var result = from n in list
                        where n.T_ID == t_id
                        select n;
        return result;
    }
}
```

在界面上将实现根据类型来查询新闻，当不选择类型(即选择“全部”)时则显示所有新闻，所以在 GetNewsByT_ID 方法中规定参数为-1 时则表示选择全部新闻。

(8) 设计界面 NewsList.aspx 页面如图 1-24 所示。



图 1-24

(9) 新闻类别和 odsType 的代码如下：

```
新闻类型：<asp:DropDownList ID="ddlType" runat="server"
 DataSourceID="odsType" DataTextField="T_Name"
 DataValueField="T_ID"
        AppendDataBoundItems="True" AutoPostBack="True">
      <asp:ListItem Value="-1">全部</asp:ListItem>
</asp:DropDownList>
<asp:ObjectDataSource ID="odsType" runat="server"
        SelectMethod="GetAllType"
        TypeName="BLL.Type"></asp:ObjectDataSource>
```

为了防止绑定时“全部”项被替换，需要设置 ddlType 的 AppendDataBoundItems 属性为 true。并且一定要设置“全部”项的 Value 属性值为-1，这样才能和 BLL 中的 GetNewsByT_ID 方法对应。

(10) GridView 和 odsNews 的代码如下：

```
<asp:ObjectDataSource ID="odsType" runat="server"
        SelectMethod="GetAllType"
```



```
    TypeName="BLL.Type"></asp:ObjectDataSource>
<asp:GridView ID="GridView1" runat="server"
    AutoGenerateColumns="False"
    DataSourceID="odsNews" DataKeyNames="N_ID">
   <Columns>
      <asp:BoundField DataField="N_ID" HeaderText="编号" />
      <asp:BoundField DataField="N_Title" HeaderText="标题" />
      <asp:BoundField DataField="N_Author" HeaderText="作者" />
      <asp:TemplateField HeaderText="所属类别" >
         <ItemTemplate>
            <asp:Label ID="Label1" runat="server"
               Text='<%# Eval("Type.T_Name") %>'></asp:Label>
         </ItemTemplate>
      </asp:TemplateField>
      <asp:TemplateField>
         <ItemTemplate>
            <asp:LinkButton ID="LinkButton1" runat="server"
               CommandName="Delete">删除</asp:LinkButton>
            <asp:HyperLink ID="HyperLink1" runat="server"
        NavigateUrl='<%#"~/NewsEdit.aspx?N_ID="+Eval("N_ID") %>'>
            详细</asp:HyperLink>
         </ItemTemplate>
      </asp:TemplateField>
   </Columns>
</asp:GridView>
<asp:ObjectDataSource ID="odsNews" runat="server"
```

```
    DataObjectTypeName="Model.News" DeleteMethod="DeleteNews"
    SelectMethod="GetNewsByT_ID" TypeName="BLL.News">
   <SelectParameters>
       <asp:ControlParameter ControlID="ddlType"   Name="t_id"
           PropertyName="SelectedValue" Type="Int32" />
   </SelectParameters>
</asp:ObjectDataSource>
```

运行网站，结果正确。

◆ 第二阶段 ◆

练习：在第一阶段练习的基础上添加 NewsEdit.aspx 页面，在该页面实现编辑和添加新闻的功能。

【问题分析】

业务逻辑层已经实现了更新和添加功能，只需要在界面上拖放一个 FormView 控件和一个 ObjectDataSource 即可。

【课后作业】

为 NewsDB 数据库创建一个用户信息表，直接在页面中使用 LINQ to SQL 实现显示、删除、更新和添加用户功能。

第2章

用户控件与HttpHandler



课程目标

- 掌握用户控件的创建
- 利用 HttpHandler 显示图片数字水印

简　介

在进行 Web 开发时通常会出现这样的情况，即可用的工具功能虽然强大，但是却不符合某一具体项目的需求。这时就需要自己动手编写控件，这种控件称为用户控件。

另外，在这一章中还将学习 HttpHandler 技术，通过对它的学习，将对 ASP.NET 的运行机制有一个更加深刻的认识，并利用该项技术实现图片数字水印。

2.1　用户控件

用户控件可用来实现页面中可重用的代码，是可以一次编写就多处方便使用的功能块。它们是 ASP.NET 控件封装最简单的形式。由于它们最简单，因此创建和使用它们也是最简单的。用户控件实际上是把已有的服务器控件组合到一个容器控件中，这样就可以创建出能在整个 Web 项目中使用的功能强大的对象了。

2.1.1　什么是用户控件

简单来说，用户控件是能够在其中放置标记和 Web 服务器控件的容器，可以被看作一个独立的单元，拥有自己的属性和方法，并可被放入到 ASPX 页面上。其工作原理与 ASP.NET 页面非常相似。也可以这样理解：当一个 Web 窗体被当作 Server 控件使用时，这个 Web 窗体便是用户控件。

如图 2-1 所示，给出了一个使用用户控件的简单例子，以便给大家更加直观的印象。在这个例子中，实现了一个能够通过单击日期控件，自动在输入框中输入日期数据的用户控件。因为 ASP.NET 没有提供能够完成这个功能的内置控件，我们只能通过用户控件来实现。





图 2-1

2.1.2　创建用户控件

在 Visual Studio 2008 中创建用户控件非常简单。要创建新的用户控件，首先应在 Web 站点中添加一个新的“Web 用户控件”文件。新建 Example2_1 网站，在网站中选择“添加新项”，打开“添加新项”对话框，在列表中选择“Web 用户控件”模板，单击“添加”按钮。注意，在将文件添加到项目后，该文件的扩展名是.ascx，如图 2-2 所示，它和创建页面一样简单。这个扩展名告诉 ASP.NET，这个文件是一个用户控件。如果试图把用户控件直接加载到浏览器中，ASP.NET 就会返回一个错误，说明这种类型的文件不能用于浏览器。

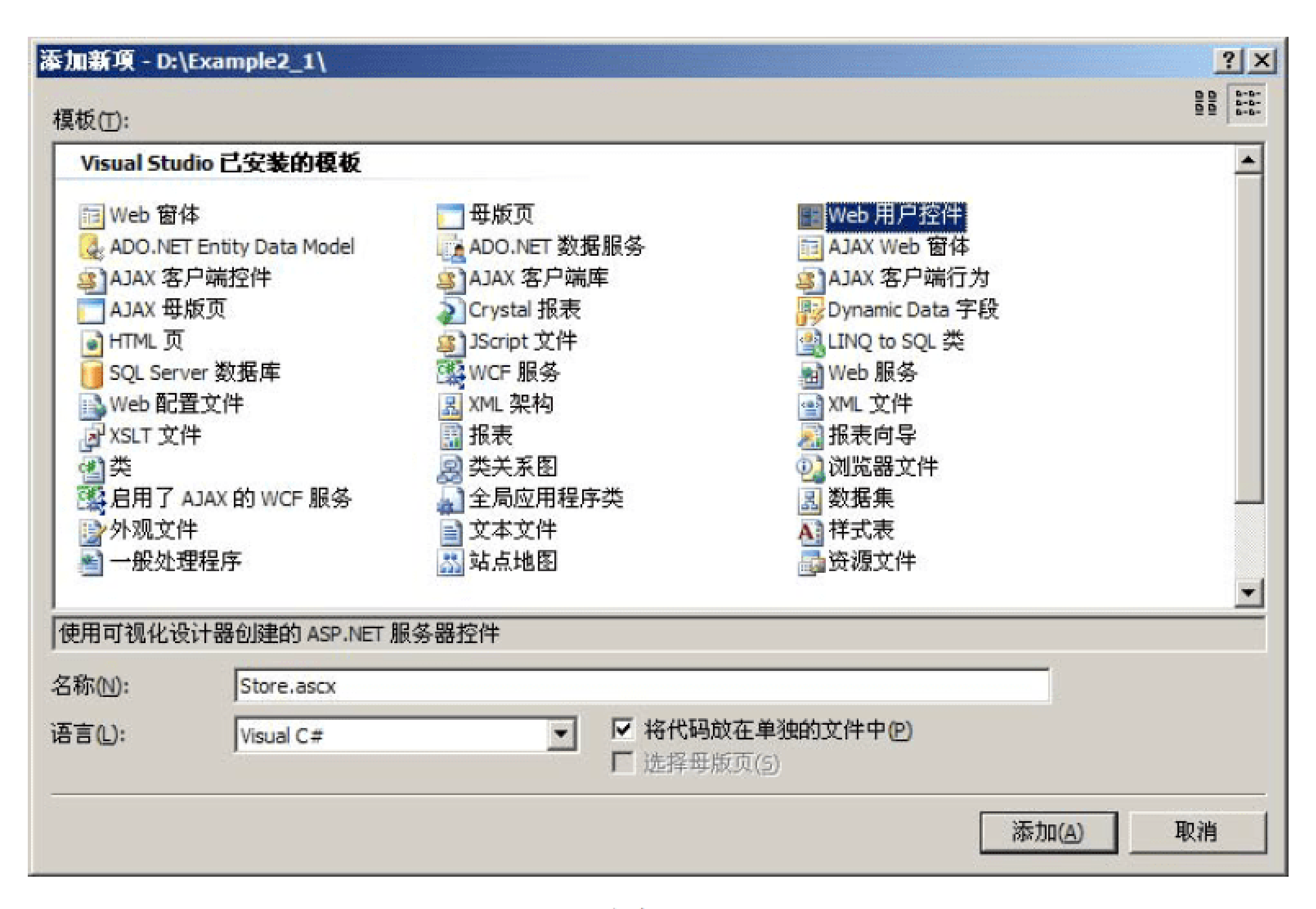


图 2-2

如果查看用户控件的 HTML 源代码，就会看到它与标准 ASP.NET Web 页面的几个有趣的区别。代码如下：

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="Store.ascx.cs" Inherits="Store" %>
```

首先注意，源代码使用@Control 指令来代替标准 Web 页面使用的@Page 指令。其次，与标准的 ASP.NET Web 页面不同，.ascx 文件只有一行代码，没有其他 HTML 标记。在一个.ascx 文件中不能包含<head>、<form>或者<body>标签，因为包含.ascx 文件的.aspx 文件已经包含了这些标签。实际上，如果试图给用户控件添加服务器端的窗体标记，在把页面传送给客户机时，ASP.NET 就会返回一个错误。该错误消息说明，Web 页面中只能有一个服务器端的窗体标记。

在 eshop 电子商务网站中有 4 个不同的商城，每个商城都需要显示一个类型列表，可以将类型列表封装成一个用户控件，这样需要显示多个商城的类型列表时，直接拖放多个用户控件即可。

为了区分不同的商城，需要在后台定义一个属性，代码如下：

```
public int StoreID { get; set; }
```

设计界面，代码如下：

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="Store.ascx.cs" Inherits="Store" %>
<table style="width: 185px">
    <tr>
        <td style="background-image: url('images/left.gif');
        background-repeat: no-repeat;width:195px;height:26px;">
                   
            <asp:HyperLink ID="hylStoreName" runat="server"
              Text=""></asp:HyperLink>
        </td>
```



```
        </tr>
        <tr>
            <td style="width:195px; ">
                <asp:Repeater ID="rptStore" runat="server"
                    DataSourceID="dsSubStore">
                    <HeaderTemplate>
                        <ol>
                    </HeaderTemplate>
                    <ItemTemplate>
                    <li><a href='<%# Eval("typeid","typies.aspx?typeid={0}") %>'>
                        <%#Eval("typename") %></a></li>
                    </ItemTemplate>
                    <FooterTemplate>
                        </ol>
                    </FooterTemplate>
                </asp:Repeater>
            </td>
        </tr>
</table>
<asp:SqlDataSource ID="dsSubStore" runat="server"
        ConnectionString="<%$ ConnectionStrings:constr %>"
        SelectCommand="SELECT * FROM [typies] WHERE ([storeID] = @storeID)">
        <SelectParameters>
            <asp:Parameter Name="storeID" Type="Int32" />
        </SelectParameters>
</asp:SqlDataSource>
```

编辑后台代码如下：

```
public partial class Store : System.Web.UI.UserControl
{
    //代表商城编号的属性
    public int Storeid { get; set; }

    protected void Page_Init(object sender, EventArgs e)
    {
        dsSubStore.SelectParameters[0].DefaultValue = Storeid.ToString();
        this.hylStoreName.Text = GetStoreName();
        this.hylStoreName.NavigateUrl = "~/Store.aspx?storeid=" + Storeid;
        this.rptStore.DataBind();
    }

    //根据商城编号获取商城名称
    public string GetStoreName()
    {
        SqlDataSource ds = new SqlDataSource();
        ds.ConnectionString = ConfigurationManager.
          ConnectionStrings["constr"].ConnectionString;
        ds.SelectCommand = "select storename from store where storeid=" + Storeid;
        DataView dv = (DataView)ds.Select(new DataSourceSelectArguments());
        if (dv.Count == 1)
        {
            return dv[0][0].ToString();
        }
```

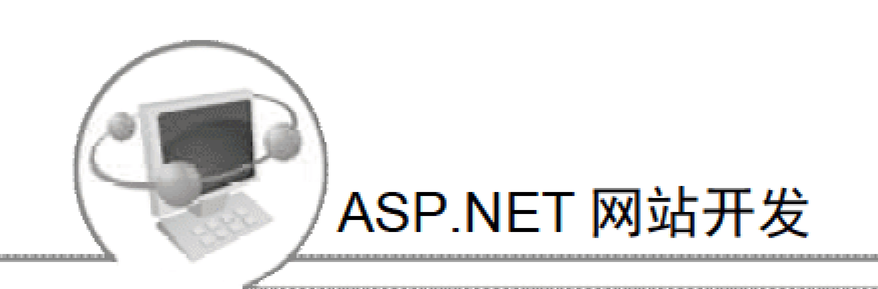


```
            else
                return null;
        }
}
```

2.1.3　使用用户控件

在 ASP.NET 中，使用用户控件也是一个非常简单的事情。要把用户控件放在标准的 ASP.NET Web 页面上，我们可以直接把用户控件文件.ascx 从“解决方案资源管理器”中拖放到需要使用的 Web 页面上，然后设置两个控件的 StoreID 属性。拖放后的页面代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>
<!--注册用户控件-->
<%@ Register Src="Store.ascx" TagName="Store" TagPrefix="uc1" %>
<head runat="server">
    <title>测试用户控件</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <uc1:Store ID="Store1" runat="server" Storeid="1" />
        <uc1:Store ID="Store2" runat="server" Storeid="2" />
    </div>
    </form>
</body>
</html>
```



把用户控件放在 Web 页面上后，在浏览器中打开该页面，就会看到完全显示出来的 Web 页面。从代码中，我们可以看到页面增加了一个@Register 指令，该指令包含以下属性。

- TagPrefix 属性：定义控件位置的命名空间。有了命名空间的制约，就可以在同一个网页里使用不同功能的同名控件。
- TagName 属性：指向所使用控件的名字。在同一个命名空间里的控件名是唯一的。控件名一般都表明控件的功能。
- Src 属性：指向控件的资源文件。资源文件使用虚拟路径("control.ascx"或"/path/control.ascx")，不能使用物理路径("C:\path\control.ascx.")。

控件注册之后，就可以像其他服务端控件一样被使用。通过定义目标前缀(TagPrefix)和目标名(TagName)，就可以像使用服务端内建控件一样进行使用。同时也确定了使用服务端运行(runat="server")方式。下面是网页调用用户控件的基本方式：

```
<uc1:Store ID="Store1" runat="server" Storeid="1" />
```

运行 Default.aspx 页面，结果如图 2-3 所示。



图 2-3



2.2　模块和处理程序

前面我们主要学习了有关控件的使用。ASP.NET 是以控件配合事件作为代码的开发方式，所以控件在 ASP.NET 的开发中尤为重要。不过在一些场合使用控件并不能达到目的，例如，在网站中我们基于著作权的保护，对站点中提供的图片要添加上网站的标识，如果只使用控件根本不可能实现，这就需要我们下面要讲的知识。

2.2.1　封面数字水印的需求

数字水印是一项应用很广的技术，那么什么是数字水印呢？下面我们看一下图 2-4。

如图 2-4 所示，在每张图片的左上角都印有新浪的图标，这就是一种典型的数字水印。数字水印技术的使用，能够很好地保护著作权，防止他人盗用受保护的资源。



图 2-4

那我们如何在图片上加上受保护的标识呢？大致有几种方法案：第一种方案我们可以使用一些图片编辑软件，对每一个原始的图片进行编辑，但是这种方法费时费力，如果要处理上千张图片，可以想象这是一件多么繁琐的事情；第二种方案可以通过编程方式给图片批量添加数字水印，这种方式不难实现，同时也解决了第一种方案费时费力的问题，但同时也带来另一个缺陷——原始图片被破坏了。有没有办法既方便又不破坏原始图片，只是在服务器发送图片到客户端前我们做一些处理，动态地添加上数字水印效果呢？当然存在了，这就是我们马上要学习的 HttpHandler 技术。

2.2.2　HttpModule 和 HttpHandler

在开始之前，我们先对 ASP.NET 的运行机制做一个更深入的学习。ASP.NET 请求处理过程是基于管道模型的，这个管道模型由多个 HttpModule 和 HttpHandler 组成，ASP.NET

把 HTTP 请求依次传递给管道中的各个 HttpModule，最终被 HttpHandler 处理，处理完成后，再次经过管道中的 HTTP 模块，把结果返回给客户端。我们可以在每个 HttpModule 中干预请求的处理过程。

在整个生命周期中，它们大致的执行过程是这样的：client 端发送页面请求，被 IIS 的某个进程截获，它根据申请的页面后缀(.aspx)不同，调用不同的页面处理程序(.asp->asp.dll；.aspx->ISAPI.dll)，而页面处理程序在处理过程中，则要经历 HttpModule，HttpHandler 的处理，前者 HttpModule 用于页面处理前和处理后的一些事件的处理，后者 HttpHandler 进行真正的页面处理。如前所说，HttpModule 会在页面处理前后对页面进行处理，所以它不会影响真正的页面请求。通常用在给每个页面的头部或者尾部添加一些信息(如版权声明)等。曾经见过一些免费的空间，我们的页面上传上去后，浏览的时候发现，在每个页面的头部和尾部多了很多小广告，如果理解了 HttpModule 的原理，要做这个就不是很难了。图 2-5 所示为 ASP.NET 处理 HTTP 请求的内部过程。

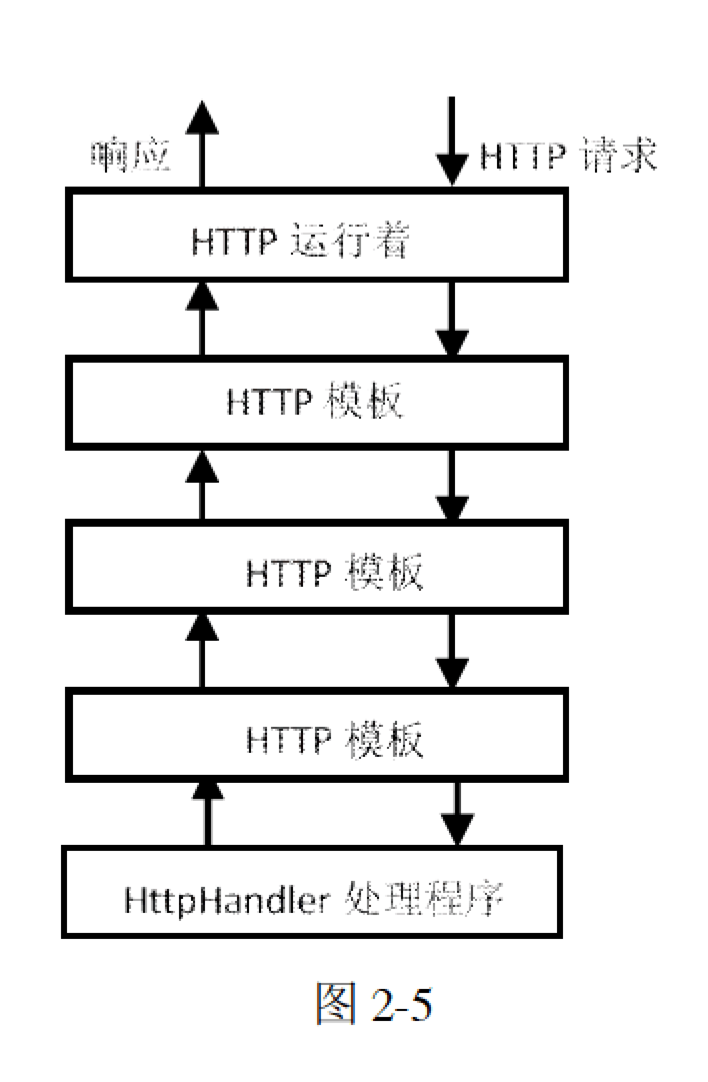


图 2-5


注意

在 HTTP 请求的处理过程中，只能调用一个 HttpHandler，但可以调用多个 HttpModule。当请求到达 HttpModule 的时候，系统还没有对这个请求真正处理，但是我们可以在这个请求传递到处理中心(HttpHandler)之前附加一些其他信息，或者对截获的这个请求做一些额外的工作，也或者终止请求等。在 HttpHandler 处理完请求之后，我们可以再在相应的 HttpModule 中把请求处理的结果进行再次加工返回客户端。

2.2.3 HttpHandler 概述

在 ASP.NET 中，我们可以很方便地创建 HttpHandler 的应用。现在使用 HttpHandler 来为 eshop 的图片增加数字水印。



我们在“添加新项”中添加“一般处理程序”，如图 2-6 所示。

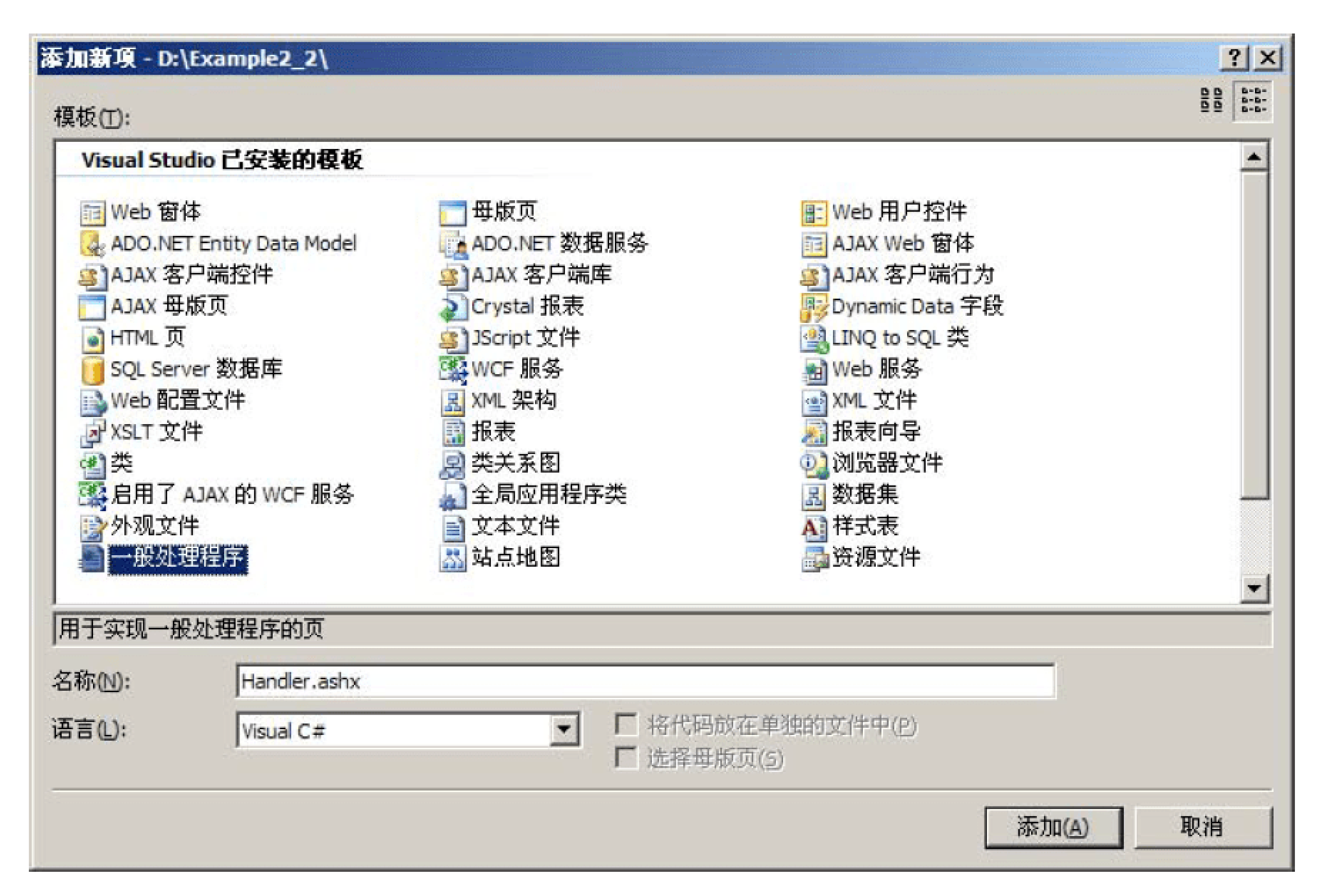


图 2-6

单击“添加”按钮后，创建了一个以.ashx 为后缀名的文件。默认创建的内容为：

```
<%@ WebHandler Language="C#" Class="Handler" %>

using System;

using System.Web;

public class Handler : IHttpHandler

{

    public void ProcessRequest(HttpContext context)

    {

        context.Response.ContentType = "text/plain";

        context.Response.Write("Hello World");

    }


    public bool IsReusable
```

```
    {
        get
        {
            return false;
        }
    }
}
```

这段代码说明 context 对象可控制输出的内容和类型。ContentType 用于设置程序的输入类型，直接访问该 HttpHandler 程序，会输出“Hello World”的文字，如图 2-7 所示。



图 2-7

从提供的代码中，我们可以看到实现 HttpHandler 的功能必须实现 IhttpHandler 接口，而且任何实现了该接口的类都可以用于处理输入的 HTTP 请求。要实现该接口，需要实现 IsReusable 属性和 ProcessRequest 方法。

其中，IsReusable 属性用于设置是否可重用该 HttpHandler 的实例；ProcessRequest 方法是整个 HTTP 请求最终的处理方法。该方法需要一个 HttpContext 类型的参数，它封装了有关个别 HTTP 请求的所有 HTTP 特定的信息。事实上，在 Page 对象中有一个 HttpContext 的实例——Context 对象。在这里，它被用于在不同的 HttpModule 和 HttpHandler 之间传递数据，也可以用于保持某个完整请求的相应信息。

2.2.4 封面数字水印的实现(指定 Handler 方式)

现在我们就可以通过.ashx 文件，实现在 eshop 图片上添加数字水印的效果了。



修改.ashx 文件提供的默认代码，源代码如下：

```
<%@ WebHandler Language="C#" Class="PicCover" %>


using System;

using System.Web;

using System.IO;

using System.Drawing;


public class PicCover : IHttpHandler

{

    //封面图片路径

    private const string OLDPICADD = "~/productimgs/";

    //默认图片的路径

    private string DEFAULTIMG = "~/productimgs/default.jpg";


    public void ProcessRequest(HttpContext context)

    {

        //获取要添加数字水印的图片物理路径

        string path = context.Request.MapPath(OLDPICADD +

                        context.Request.Params["id"].ToString() + ".jpg");

        System.Drawing.Image Cover;

        //判断相应物理路径中的文件是否存在，不存在则使用默认图片

        if (File.Exists(path))

        {

            //加载文件

            Cover = Image.FromFile(path);
```

```
            //实例化画布
            Graphics g = Graphics.FromImage(Cover);
            //创建字体
            Font font = new Font("新宋体", 20);
            //在 Image 上绘制水印
            g.DrawString("eshop", font, Brushes.Red, 0, Cover.Height - font.Height);
            //释放画布
            g.Dispose();
        }
        else
        {
            Cover = Image.FromFile(context.Request.MapPath(DEFAULTIMG));
        }

        //设置输出类型为 JPEG 图片
        context.Response.ContentType = "image/jpeg";
        //将修改的图片存入输出流
        Cover.Save(context.Response.OutputStream, System.Drawing.Imaging.ImageFormat.Jpeg);
        Cover.Dispose();
        context.Response.End();
    }
    public bool IsReusable
    {
        get
        {
            return false;
```



```
        }
    }
}
```

我们还要做的就是，将所有需要使用数字水印访问图片的路径修改为“PicCover.ashx?id=数字”就可以了，这时我们就可以看到封面图片的左下角添加上了“eshop”的标识，完成了数字水印的效果。接着我们打开 Image 文件夹查看封面图片的原图，发现原图没有做任何的修改。真是太神奇了！

2.2.5　数字水印的实现(全局 Handler 方式)

上述这种做法仍然有点缺憾：还需要把访问封面图片的路径都修改掉。现在换一种方法，可以在不修改任何访问路径的情况下实现图片的数字水印效果。这就只能通过对捕获的图片的访问请求来实现，需要做如下处理：

- 修改 web.config，将所有对.jpg 内容的访问转到 HttpHandler 处理程序。
- 获得访问请求，得到用户访问的图片路径。
- 根据请求的路径查找相对应的封面图片。
- 将网站标识在封面图片的左下角输出。
- 修改程序的输出类型，并将组合出的新图片输出。

可以看出和前面的方法基本上是一致的，只是做了一个配置，捕获用户针对 jpg 格式图片的请求就行了。

首先我们修改 Web.config 文件，在文件中添加以下代码行：

```
<httpHandlers>
    <add verb="*" path="productimgs/*.jpg" type="PicCoverHandler"/>
</httpHandlers>
```

- verb：指定谓词列表可以是逗号分隔的 HTTP 谓词列表(例如，"GET, PUT, POST")。[*]通配符，此处标识所有的请求。

- path：指定路径属性可以包含单个 URL 路径或简单的通配符字符串(如*.aspx)。
- type：指定逗号分隔的类/程序集组合。ASP.NET 首先在应用程序的专用\bin 目录中搜索程序集 dll，然后在系统程序集缓存中搜索程序集 dll。

接着我们创建一个实现了 IhttpHandler 的类 PicCoverHandler，此类的代码与上面方法的代码非常相似，只不过我们不再需要获取 DVD 的 Id 值了，代码如下：

```
<%@ WebHandler Language="C#" Class="PicCoverHandler" %>


using System;
using System.Web;
using System.IO;
using System.Drawing;


public class PicCoverHandler : IHttpHandler
{
    //默认图片的路径
    private string DEFAULTIMG = "~/productimgs/default.jpg";
    public void ProcessRequest(HttpContext context)
    {
        System.Drawing.Image Cover;
        //判断相应物理路径中的文件是否存在，不存在则使用默认图片
        if (File.Exists(context.Request.PhysicalPath))
        {
            //加载文件
            Cover = Image.FromFile(context.Request.PhysicalPath);
```



```
            //实例化画布
            Graphics g = Graphics.FromImage(Cover);
            //创建字体
            Font font = new Font("新宋体", 20);
            //在 Image 上绘制水印
            g.DrawString("eshop", font, Brushes.Red, 0, Cover.Height - font.Height);
            //释放画布
            g.Dispose();
        }
        else
        {
            Cover = Image.FromFile(context.Request.MapPath(DEFAULTIMG));
        }

        //设置输出类型为 JPEG 图片
        context.Response.ContentType = "image/jpeg";
        //将修改的图片存入输出流
        Cover.Save(context.Response.OutputStream, System.Drawing.Imaging.ImageFormat.Jpeg);
        Cover.Dispose();
        context.Response.End();
    }

    public bool IsReusable
    {
```

```
            get
            {
                return false;
            }
        }
}
```

直接运行程序，结果如图 2-8 所示，达到预期的效果。

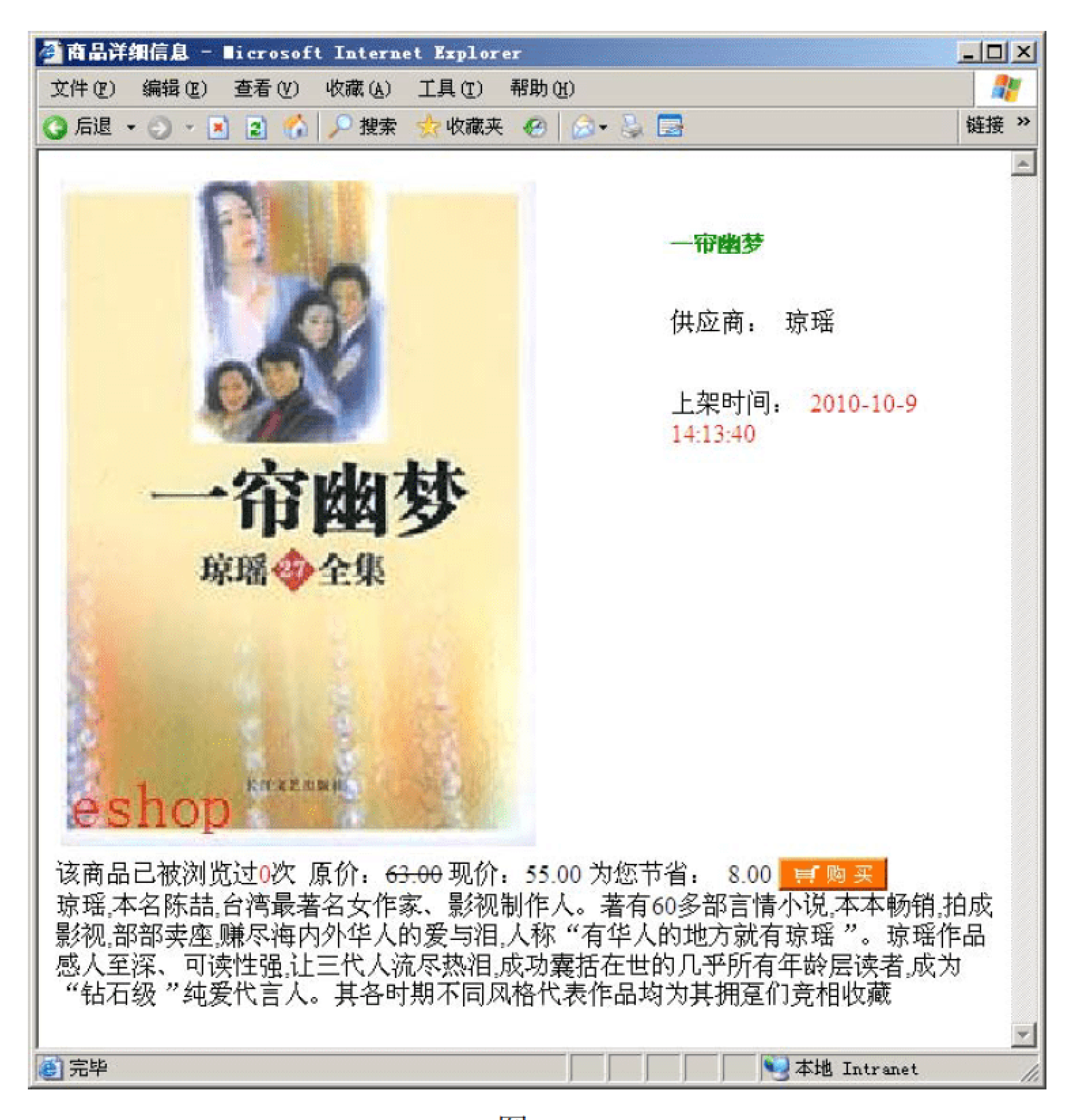


图 2-8

需要注意的是，使用这种配置方式，在开发服务器上运行时没有任何问题，可是当我们将网站部署到 IIS 上运行时，会发现没有任何的效果。是什么原因造成的呢？原来，我们在开发调试时使用 Visual Studio 2008 提供的开发服务器，它仅提供了最简单的 Web 服务器功能，它不会对请求的内容做任何的处理，而直接将所有的请求转交给 ASP.NET 处



理。相比而言，IIS 却是一个比较完善、功能强大的 Web 服务器。我们所有提交到 IIS 的请求，都会依据后缀名而做一些分类处理。默认情况下，.html、.jpg 等静态格式的文件 IIS 不会做任何处理而直接将结果返回。只有当后缀名符合相关条件时(如.aspx、asp)，才会将请求转交给相应的处理程序。

所以不难看出，不是我们的程序有问题，在 IIS 上不能正常工作，而是 IIS 根本没有将我们的请求转交给 ASP.NET，那我们写的程序当然就没有被执行，也就出不来任何的效果了。问题找到了，那我们就相应地要进行以下操作，让 IIS 将我们对.jpg 的请求转交给 ASP.NET 进行处理。

(1) 打开 IIS 控制台管理程序，选中对应的站点目录，打开它的属性对话框，如图 2-9 所示。

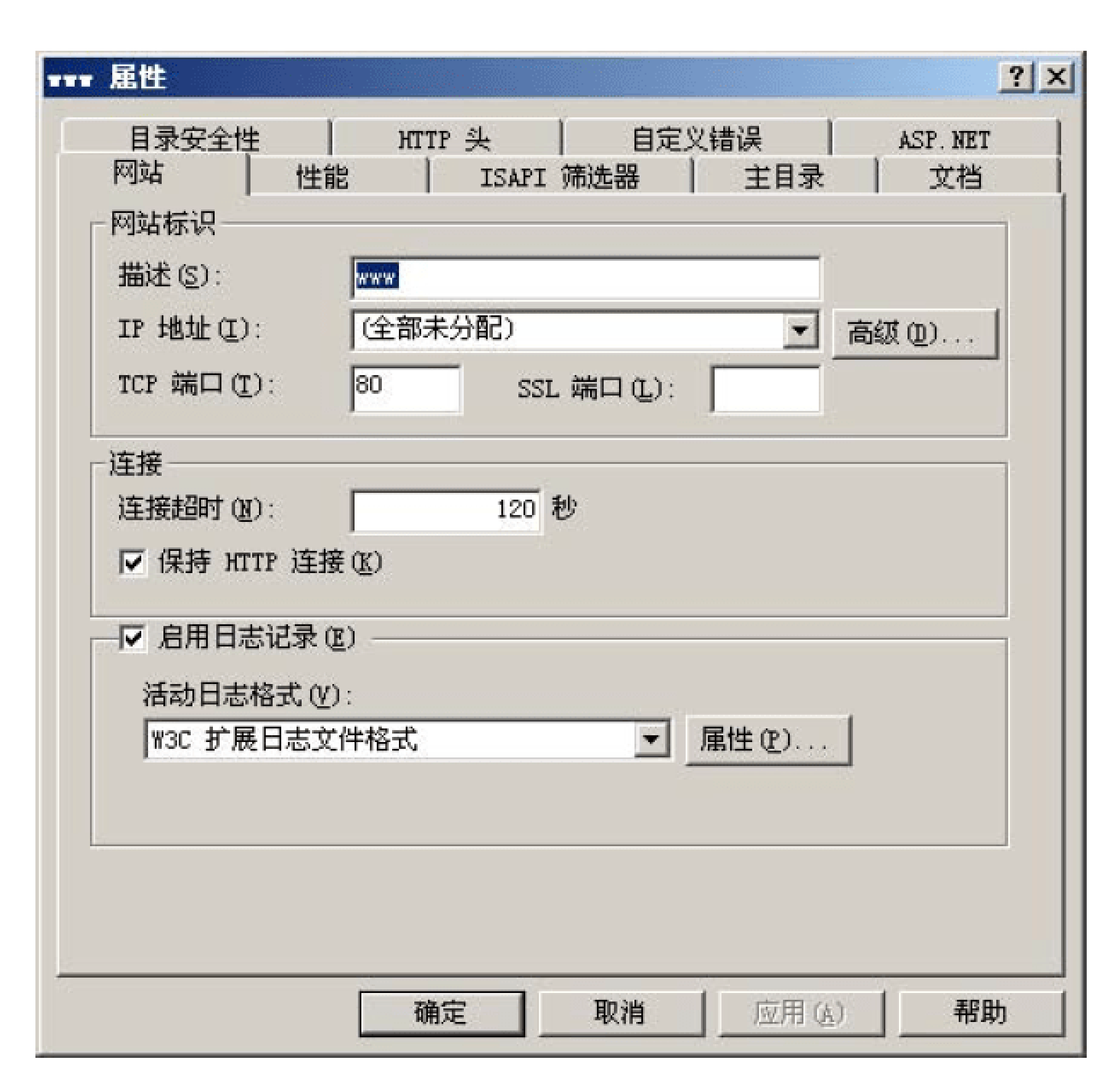


图 2-9

(2) 在该属性对话框的“主目录”选项卡中，单击“配置”按钮，打开“应用程序配置”对话框，如图 2-10。

(3) 在“应用程序配置”对话框的“映射”选项卡中单击“添加”按钮，打开“添加/编辑应用程序扩展名映射”对话框，如图 2-11 所示。



图 2-10



图 2-11

注意要选择当前版本的 aspnet_isapi.dll。

【小结】

- 使用用户控件简化复杂开发。
- 能利用 HttpHandler 显示图片数字水印。



【自测题】

1. 在使用用户控件的页面中用(　　)指令来注册用户控件。

   A. Regedit　　B. Regsvr32　　C. Reg　　D. Register

2. 下列不属于用户控件特性的是(　　)。

   A. 以.ascx 为扩展名可在 ASP.NET 布局代码中可重用的部分

   B. 不可以包含<Form>标记

   C. 使用@Control 指令

   D. 可以直接使用用户控件

3. 用户控件的@Control 指令表明所生成的类是从(　　)继承。

   A. System.Web.UI.UserControl　　B. System.Web.UI.HtmlControl

   C. System.Web.UI.WebControl　　D. System.Web.UI

4. 关于 HttpHandler 程序的说法正确的是(　　)。

   A. 它是请求处理的终点

   B. 它实现了 IhttpHandler 接口

   C. IsReusable 属性设置为 false 时，表明程序只能使用一次

   D. 文件后缀名是.ashx

5. 用户控件的扩展名是(　　)。

   A. .asax　　B. .asmx　　C. .aspx　　D. .ascx

【上机部分】

上机目标

- 使用 HttpHandler

- 使用 HttpModule

上机练习

◆ 第一阶段 ◆

练习 1：阻止图片盗链。

【问题描述】

什么是图片盗链呢？让我们先分析下一般的浏览现象，其中最重要的一点就是一个完整的页面并不是一次全部传送到客户端的。如果你请求的是一个带有许多图片和其他信息的页面，那么最先的一个 HTTP 请求被传送回来的是这个页面的文本，然后通过客户端的浏览器对这段文本的解释执行，发现其中还有图片，那么客户端的浏览器会再发送一条 HTTP 请求，当这个请求被处理后这个图片文件会被传送到客户端，然后浏览器会将图片安放到页面的正确位置，就这样一个完整的页面也许要经过发送多条 HTTP 请求才能够被完整地显示。基于这样的机制，就会产生一个问题，那就是盗链问题：就是一个网站中如果没有页面中所说的信息，例如图片信息，那么它完全可以将这个图片连接到别的网站。这样没有任何资源的网站利用了别的网站的资源来展示给浏览者，提高了自己的访问量，而大部分浏览者又不会很容易地发现，显然，这对于那个被利用了资源的网站是不公平的。

【问题分析】

现在利用 ASP.NET 中的 HttpHandler 能够很好地解决这个问题，之所以能够发生这个问题，就是因为我们在默认状态下只处理那些动态的网页，像 asp、aspx 等，但当有请求一个图片文件时，IIS 就会直接提取资源并发送给客户端，这样看来就显得有些盲目了吧，所以我们要创建自己的 HttpHandler 来处理图片文件。

【参考步骤】

(1) 启动 Visual Studio 2008，新建网站。

(2) 添加新项，创建一个“一般处理程序”。在 ProcessRequest 方法中加入对请求图片文件的 HTTP 请求进行处理的代码。完整代码如下：



```
<%@ WebHandler Language="C#" Class="Handler" %>

using System;
using System.Web;

public class Handler : IHttpHandler {

    public void ProcessRequest(System.Web.HttpContext context)
    {
        //判断是否是本地引用，如果是则返回给客户端正确的图片
        //这里的判断就是用到了 HTTP 请求中所记录的参考页信息
        if(context.Request.UrlReferrer.Host == "localhost")
        {
            //设置客户端缓冲中文件过期时间为 0，即立即过期
            context.Response.Expires = 0;
            //清空服务器端为此会话开辟的输出缓存
            context.Response.Clear();
            //将请求文件写入到服务器端为此会话开辟的输出缓存中
            context.Response.WriteFile(context.Request.PhysicalPath);
            //将服务器端为此会话开辟的输出缓存中的信息传送到客户端
            context.Response.End();
        }
        else //如果不是本地引用，则属于盗链引用，返回给客户端错误的图片
        {
            context.Response.Expires = 0;
            context.Response.Clear();
```

```
                context.Response.ContentType = "image/jpeg";
                //将特殊的报告错误的图片文件写入到服务器端为此会话开辟的输出缓存中
                context.Response.WriteFile("error.jpg");
                context.Response.End();
            }
        }

        public bool IsReusable {
            get {
                return false;
            }
        }
}
```

(3) 在 Web.config 应用程序配置文件中加入注册信息，注册自定义的 HttpHandler。代码如下：

```
<httpHandlers>
<add verb = "*" path = "*.jpg" type = "Handler" />
</httpHandlers>
```

练习 2：为 eshop 实现 foot 用户控件。

【问题描述】

只提供导航信息和说明文字，界面如图 2-12 所示。

图 2-12



【参考步骤】

(1) 打开 eshop 网站。

(2) 添加用户控件 foot.ascx，并添加设计代码如下：

```
%@ Control Language="C#" AutoEventWireup="true" CodeFile="foot.ascx.cs" Inherits="inc_foot" %>
<div class="foot"> | <a
onclick="this.style.behavior='url(#default#homepage)';this.setHomePage('http://www.eshop.com');"
href="http://www.eshop.com/">设为首页</a> |
<a href="javascript:window.external.AddFavorite  'http://www.eshop.com/','电子商务')">加入收藏</a> |
<a href="mailto:sales@mxctech.com">联系我们</a> |
<a href="/admin/login.aspx" target="_blank">网站维护</a> |</div>
<div class="copyright">
    南湖电子商务有限公司　版权所有 <br>
    Copyright &copy; 2006 - 2011 All Rights Reserved   邮编：430000 <br>
    地址:武汉市洪山区南湖山庄   
    业务电话：027-88427799<br>
    服务信箱：<a href="mailto:sales@mxctech.com">sales@mxctech.com</a>
</div>
```

◆ 第二阶段 ◆

练习 1：利用 HttpModule 进行身份检查。

【问题描述】

大家在进行登录时，登录成功后，一般要把用户名放在 Session 中保存，在其他每一个页面的 Page_Load 事件中都检查 Session 中是否存在用户名，如果不存在就说明用户未登录，就不让其访问其中的内容。在比较大的程序中，这种做法实在是太笨拙，因为你几乎要在每一个页面中都加入检测 Session 的代码，导致难以开发和维护。下面我们看看如

何使用 HttpModule 来减少工作量。

【问题分析】

由于在这里要用到 Session 中的内容，我们只能在 AcquireRequestState 和 PreRequestHandlerExecute 事件中编写代码，因为在 HttpModule 中只有在这两个事件中可以访问 Session。这里我们选择 PreRequestHandlerExecute 事件编写代码。

【参考步骤】

(1) 编写一个类，实现 IhttpModule 接口。代码如下：

```
public class AuthenticModule:IHttpModule
{
    public AuthenticModule()
  {
      //
      //TODO: 在此处添加构造函数逻辑
      //
  }

    #region IHttpModule 成员

    public void Dispose()
    {
        throw new NotImplementedException();
    }

    public void Init(HttpApplication context)
    {
    }
```



```
    #endregion
}
```

(2) 在 Init 事件中注册 PreRequestHandlerExecute 事件，并实现事件处理方法。

```
public class AuthenticModule:IHttpModule
{
    public AuthenticModule()
  {
      //
      //TODO: 在此处添加构造函数逻辑
      //
  }

    #region IHttpModule 成员

    public void Dispose()
    {
        throw new NotImplementedException();
    }

    public void Init(HttpApplication context)
    {
        context.PreRequestHandlerExecute += new EventHandler
              (context_PreRequestHandlerExecute);
    }
```

```
    void context_PreRequestHandlerExecute(object sender, EventArgs e)
    {
        HttpApplication ha = (HttpApplication)sender;
        string path = ha.Context.Request.Url.ToString();
        int n = path.IndexOf("Login.aspx");
        if (n == -1) //是否是登录页
        {
//是否 Session 中有用户名，若是空的话，转向登录页
            if (ha.Context.Session["user"] == null)
            {
                ha.Context.Response.Redirect("Login.aspx?source=" + path);
            }
        }
    }
    #endregion
}
```

(3) 在 Login.aspx 页面的“登录”按钮中加入下面的代码:

```
protected void btnLogin_Click(object sender, EventArgs e)
{
    //判断用户名和密码是否正确
    if (true)
    {
        if (Request.QueryString["source"] != null)
        {
            //取出从哪个页面转来的
            string ReqPageAddress = Request.QueryString["source"].ToString();
```



```
                Session["user"] = txtUID.Text;
                //转到用户想去的页面
                Response.Redirect(ReqPageAddress);
            }
            else
            {
                //默认转向 main.aspx
                Response.Redirect("main.aspx");
            }
        }
}
```

(4) 在 Web.Conofig 中注册 HttpModule 模块。

```
<httpModules>
    <add name="TestModule" type="AuthenticModule"/>
</httpModules>
```

练习 2：为 eshop 实现 head 用户控件。

【问题描述】

提供按商城和商品名关键字搜索功能，提供注册和登录超链接，提供查看购物车超链接，显示所有商城超链接，显示当前客户姓名，提供查看当前客户的账户、订单和注销功能，界面如图 2-13 所示。

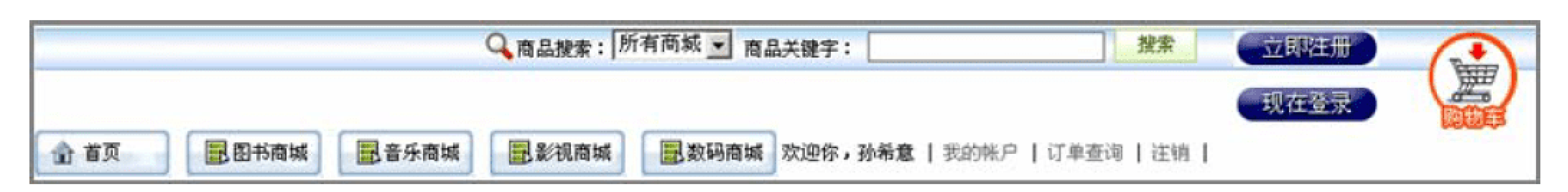


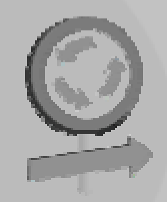
图 2-13

【课后作业】

1. 修改 eshop，创建能实现商品搜索功能的用户控件。
2. 利用 HttpHandler，修改 eshop 客户登录页面，添加验证码功能。

第3章

成员资格和角色管理



课程目标

- ▶ 理解认证和授权机制
- ▶ 掌握窗体身份认证
- ▶ 应用成员资格管理
- ▶ 应用角色管理
- ▶ 使用用户管理控件

简　介

对于大多数的 Web 应用程序而言，身份验证和授权管理是非常重要的部分。在本章我们将了解和掌握 ASP.NET 的身份验证，并学习它的授权管理。

通常的身份验证需要我们自己编写代码来检验用户名和密码是否合法，在 ASP.NET 中可以直接利用系统给出的解决方案来实现非常轻松的身份验证。

通常的授权管理我们可以通过自行创建用户表和权限表，然后编写相应的逻辑代码来完成。但是这种做法编写的代码复杂繁琐，而且在实现中还会由于考虑不周，容易出现这样或那样的漏洞和错误。而不同的系统它们有关成员验证、管理等内容都大同小异，没有本质性差别。为了解决以上矛盾，ASP.NET 提出了自己的解决方案。该解决方案主要包括 3 个部分:

(1) 提供登录系列控件。

ASP.NET 将应用程序中有关用户验证和管理的界面集成为多个登录控件。

(2) 提供 Web 站点管理工具。

ASP.NET 为每个应用程序都内建了一个特殊的 Web 管理站点。开发人员可以利用该站点轻松实现添加更新用户信息、定义角色、设置验证类型等功能。

(3) 实现成员和角色管理功能。

成员和角色管理功能能够帮助开发人员快速实现业务逻辑和数据访问。

3.1　ASP.NET 的安全模式

根据所请求资源的类型，IIS 能够自己处理请求，也可以不自己处理请求。如果资源请求一个 ASPX 页面，则 IIS 将请求经过身份验证用户(或匿名用户)的安全令牌一起传递给 ASP.NET。接下来要发生的事情就取决于 ASP.NET 的配置。

ASP.NET 支持 3 种授权方法：Windows，Passport 和 Form。还有第四种可能的方法是



None。表示 ASP.NET 自己根本不执行身份验证，完全依赖 IIS 身份验证。在这种情况下，匿名用户可以使用默认的 ASP.NET 账户访问资源。

3.1.1　Windows 身份验证

基于 Windows 的身份验证在 ASP.NET 应用程序所在的 Windows 服务器和客户机之间处理。在基于 Windows 的身份验证模型中，请求直接发送给 IIS，进行验证过程。这种类型的身份验证在内联网环境中非常有用。在该环境下，可以让服务器处理这个验证过程，尤其是用户已登录到网络上时，只需获取并利用已有的证书完成验证过程。

IIS 首先从域登录中获得用户的证书。如果这个过程失败，IIS 就显示一个弹出对话框，用户可以在其中输入或重新输入登录信息。要让 ASP.NET 应用程序使用基于 Windows 的身份验证，首先要创建一些用户和组。

3.1.2　Passport 身份验证

验证终端用户的另一种方法是使用 Microsoft 的 Passport 标识系统。有 Passport 账户的用户可以有一个签名解决方案，也就是说，他只需这些证书就可以登录到 Internet 的站点或其他支持 Passport 的站点和应用程序上。

应用程序支持 Passport 身份验证时，请求会被重定向到 Microsoft Passport 站点上，在该站点上，用户可以输入证书。如果验证成功，用户就可以获得授权，请求被重定向回应用程序。

护照身份验证是一个 Microsoft 集中式身份验证服务。护照提供一种对参与某种计划的所有站点的用户进行身份验证的方法。用户只需要登录一次，如果成功通过了身份验证，就可以自由遍历所有的成员站点。除了一次性登录服务外，护照还为成员站点提供了核心的配置文件服务。

登录服务器使用窗体信息来验证用户的身份，如果成功，则创建一个 Passport 票据。接着，用户被重定向到原始的 URL，并且该票据被加密后通过查询字符串进行传递。最后，浏览器按照重定向指令，再次请求原始的 Passport 保护的资源，然而，这时该请求包含一

个有效的票据，使用 PassportAuthenticationModule 将允许请求通过。

3.1.3 窗体身份验证

Windows 和 Passport 身份验证对于现实的 Internet 应用程序几乎都不适用。Windows 身份验证是不切实际的，因为 Web 应用程序用户必须具有该应用程序的域中的 Windows 账户。对于 Passport 身份验证也可以得出相同的结论，因为 Passport 并不是免费的，而且安全性也有待验证。

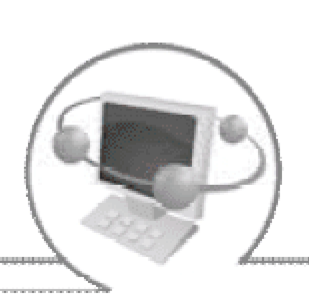
因此，实际的 Web 开发人员理想的身份验证机制是什么呢？Web 编程最佳实践建议我们把一些比较样板化的代码置于每个非公共页面上，并把用户重定向到一个登录页面。在该登录页面上，提示用户输入凭据，如果成功地通过了身份验证，则把它重定向到最初的请求页面。

窗体身份验证仅仅是 ASP.NET 内置的基础结构，用以实现上述登录模式。例如：依靠一个用户账户数据库，窗体身份验证是理想的选择。通过窗体身份验证实现的登录模式看上去与 Windows 和 Passport 身份验证没有什么根本的区别。关键区别是，使用窗体身份验证时，一切都在应用程序的严格控制下发生。

通过调整应用程序的根目录下的 Web.Config 文件，可以为窗体身份验证建立一个 ASP.NET 应用程序。

示例 3-1：

```
<system.web>
<authentication mode="Forms">
      <forms name=".ASPXAUTH" loginUrl="Login.aspx" path="/" />
      </authentication>
      <authorization>
          <deny users="?"/>
      </authorization>
</system.web>
```



<authentication>节指示用户定义的登录窗体的 URL。ASP.NET 仅向<authorization>节被显示拒绝访问的用户显示该窗体。符号“？”表示任何匿名的、未经身份验证的用户。

3.2　基于窗体的身份授权模式

基于窗体的身份验证是允许用户访问整个应用程序或其特定资源的一种流行模式。使用它可以把登录窗体直接放在应用程序中，这样，终端用户只需把用户名和密码输入到浏览器中的一个 HTML 窗体上即可。

IIS 接收请求，并且由于已启用 IIS 匿名访问，IIS 不执行任何用户身份验证，而将请求传递给 ASP.NET 应用程序。由于 ASP.NET 身份验证模式设置为 Forms，ASP.NET 应用程序将检查 Forms 身份验证票的请求(特定 Cookie)。如果没有连接到请求的身份验证票，则 ASP.NET 将请求重定向到应用程序的配置文件中指定的登录页。用户在登录页上输入所需凭据(通常为用户名和密码)。应用程序代码检查凭据以确认它们的真实性。如果凭据通过身份验证，应用程序代码将身份验证票连接到表示用户凭据的响应(不包括密码)。如果身份验证失败，通常将响应连同“访问被拒绝”消息一起返回，或重新显示登录窗体。ASP.NET 使用消息身份验证检查(MAC)来检查身份验证票的有效性。如果用户通过身份验证，ASP.NET 检查身份验证并允许对初始请求的资源的访问、将请求重定向到某个其他页或将请求重定向到自定义授权模块(其中将测试要访问受保护的资源的凭据是否已得到授权)。如果授权失败，则 ASP.NET 将用户重定向到登录页。如果用户已被授权，则将允许用户访问受保护资源，或者应用程序会在授权访问受保护资源之前要求额外进行凭据测试，具体取决于应用程序的设计。

示例 3-1 的演示代码读者已经看到了，那是基于窗体验证的配置信息。这个结构必须应用于 Web.Config 文件。首先使用配置节<authorization>元素，可以拒绝所有的匿名用户访问应用程序。只有验证用户才能访问应用程序包含的页面。

如果请求者未通过验证，就执行<authentication>元素中定义的内容。Mode 属性的值设

置为 Forms，表示 Web 应用程序使用基于窗体的身份验证。下一个指定的属性是 LoginUrl，它指向包含应用程序的登录窗体页面。在示例 3-1 中，如果试图访问应用程序的终端用户未通过身份验证，他的请求就会被重定向到 Login.aspx，以便对该用户进行验证和授权。在提供了有效的证书后，用户就会返回到他在应用程序中最初发出请求的地方。最后使用的属性 path，它指定了保存 cookie 的位置。该 cookie 用于存储授权用户的访问令牌。在大多数情况下，该值默认为“/”。图 3-1 所示是用户身份验证流程。表 3-1 列出了<forms>元素的属性。

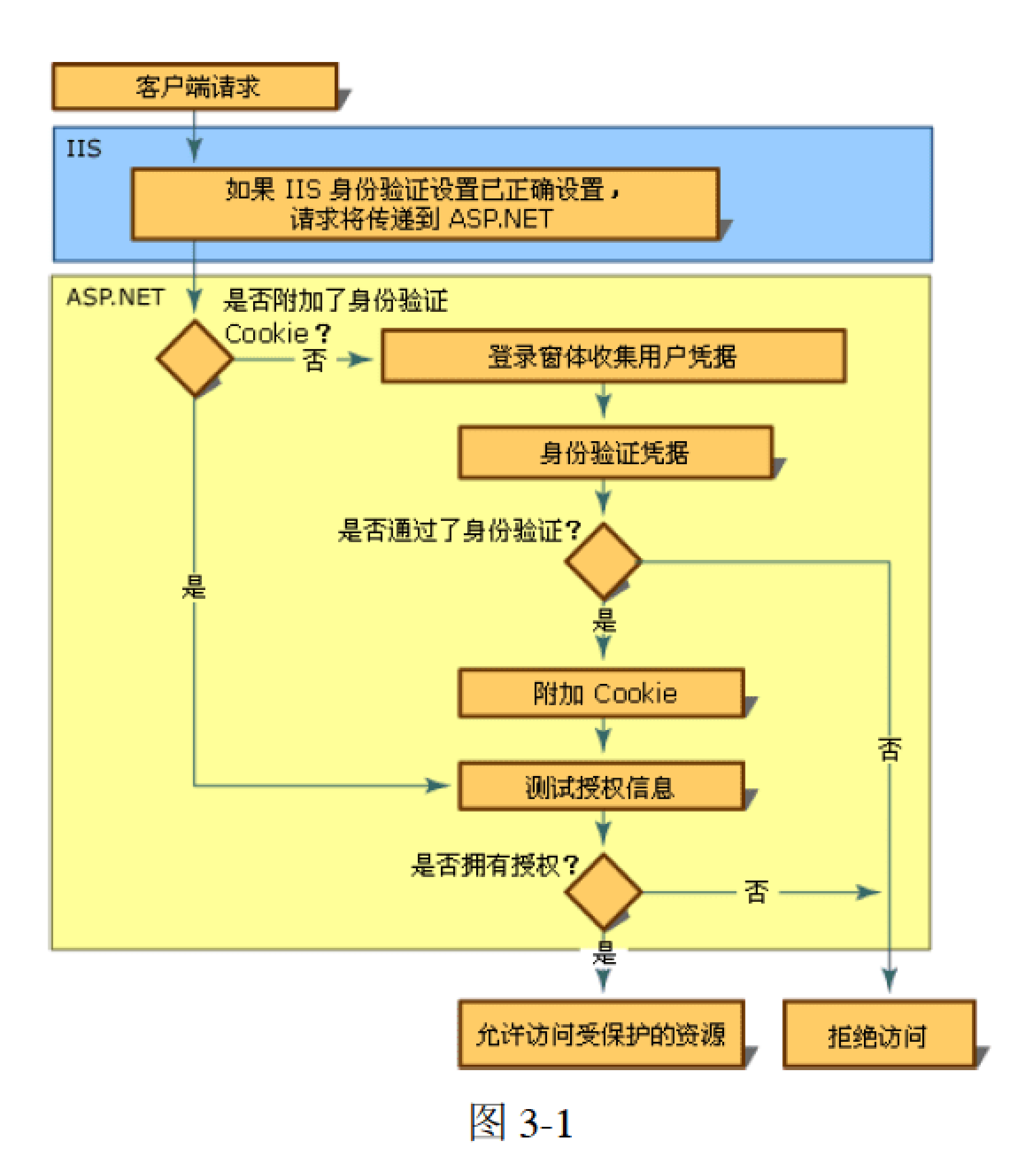


图 3-1

表 3-1　<forms>元素的主要属性

属　性	说　明
name	这是赋予 cookie 的名字，该 cookie 用于在请求之间保存用户。该默认值是.ASPXAUTH
loginUrl	如果没有找到有效的验证 cookie，就指定请求重定向的 URL
protection	指定要应用于验证 cookie 的保护级别，它有以下 4 个设置： • All：应用程序使用数据有效性验证和加密机制来保护 cookie，这是默认设置 • None：不加密 cookie • Encryption：加密 cookie，但不对它进行数据有效性验证 • Validation：进行数据有效性验证，但不加密 cookie



（续表）

属　性	说　明
path	指定应用程序所存储 cookie 的路径。在大多数情况下应用“/”，它是默认设置
timeout	指定 cookie 过期的时间(分钟)，其默认值为 30 分钟
cookieless	指定在进行验证和授权过程中，基于窗体的身份验证过程是否使用 cookie
defaultUrl	指定登录成功后默认跳转的 URL
domain	指定要与窗体身份验证一起发送的域名

下面我们演示基于窗体的身份验证。具体操作步骤如下。

(1) 新建网站 Example3_1，建立如图 3-2 所示的目录结构。

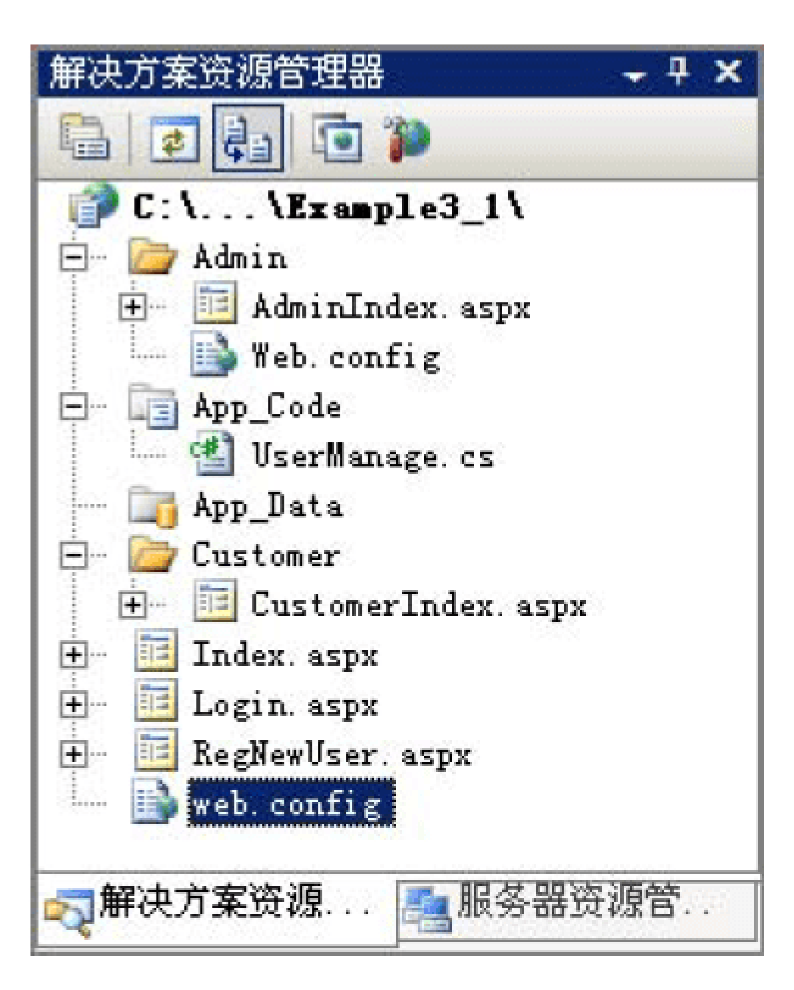


图 3-2

(2) 在 Login.aspx 中，添加“登录”按钮的单击事件。代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.Security;
```

```
public partial class Login : System.Web.UI.Page
{
    protected void btnLogin_Click(object sender, EventArgs e)
    {
        string uname = this.txtLoginId.Text;
        string upwd = this.txtLoginPwd.Text;
        if (UserManage.Login(uname, upwd))
        {
            FormsAuthentication.RedirectFromLoginPage(uname, false);
        }
        else
        {
            ClientScript.RegisterStartupScript(this.GetType(),
                "loginerr", "alert('用户名或密码不正确，请重新填写');", true);
        }
    }
}
```

(3) 修改 Web.config 文件，加入以下代码：

```
<system.web>
    <authentication mode="Forms">
        <forms loginUrl="Login.aspx" defaultUrl="Index.aspx">
    </authentication>
    <authorization>
       <deny users="?"/>
```



```
    </authorization>
</system.web>
```

(4) 运行程序，我们发现通过这几步的操作，就完成了简单的基于窗体的身份验证。

在上面的代码中，“登录”按钮的单击事件里使用了 FormsAuthentication 类的 RedirectFromLoginPage 方法，这个方法把请求从 Login.aspx 重定向到最初请求的资源。

RedirectFromLoginPage 方法有两个参数。第一个参数是用户名，用于进行 cookie 验证。这个参数并不映射为账户名，而是由 ASP.NET 的 URL 授权功能使用。第二个参数指定是否生成一个永久的 cookie。如果它设置为 True，终端用户从一个浏览器会话到下一个会话时，就不需要再次登录。

在 Example3_1 中，我们对其他页面的每个请求都会导致 ASP.NET 检查是否存在正确的身份验证令牌。如果没有找到该令牌，请求就重定向到指定的登录页面 Login.aspx，查看浏览器中的 URL 会发现，ASP.NET 使用一个查询字符串值，以确定用户获得授权后把用户返回到什么地方。

```
http://localhost:2960/Login.aspx?ReturnUrl=%2fCustomer%2fCustomerIndex.aspx
```

其中的查询字符串 ReturnUrl 使用了最初请求的页面和文件夹的值。程序根据输入的用户名和密码，在数据库中进行比对，如果正确，就调用 RedirectFromLoginPage 方法，否则就使用 RegisterStartupScript 方法，弹出一个出错提示对话框。

其实我们还可以将用户名和密码保存在 Web.config 文件中，用来检查用户名和密码是否来自授权用户。Web.config 文件的<forms>子元素还可以有子元素。子元素<credentials>允许直接在 web.config 文件中指定用户名和密码组合。可以用两种方式添加这些值。最简单的方法代码如下：

```
<authentication mode="Forms">
  <forms loginUrl="Login.aspx" defaultUrl="Index.aspx">
    <credentials passwordFormat="Clear">
```

```
        <user name="admin" password="123456"/>
        <user name="sysadmin" password="654321"/>
      </credentials>
    </forms>
</authentication>
<authorization>
    <deny users="?"/>
</authorization>
```

<credentials>元素在配置文件中添加了用户及其密码。<credentials>有一个属性 passwordFormat，其值可以是 Clear、MD5 和 SHA1。下面描述了这些选项。

- Clear：密码存储为明文。用户的密码直接与这个值比较，不需要进一步转换。
- MD5：密码使用散列摘要进行存储。在验证证书时，用户密码使用 MD5 算法进行散列，再与这个值进行相等比较。不会存储或比较明文密码。这个算法比 SHA1 的性能好。
- SHA1：密码使用 SHA1 散列摘要来存储。在验证证书时，用户密码使用 SHA1 算法进行散列，再与这个值进行相等比较。不会存储或比较明文密码。这个算法的安全性最高。

在上面的例子中，我们使用了 Clear 设置，这不是安全的方法，但可用于演示。<credentials>的一个子元素是<user>，在该子元素中，可以使用属性 name 和 password 为授权用户定义用户名和密码。

由于我们使用了<credentials>元素用来保存授权用户名和密码，那么相应的“登录”按钮单击事件中，用来验证用户名和密码的代码也应该相应地发生改变。验证代码如下：

```
protected void btnLogin_Click(object sender, EventArgs e)
{
    string uname = this.txtLoginId.Text;
```



```
    string upwd = this.txtLoginPwd.Text;
    if (FormsAuthentication.Authenticate(uname, upwd))
    {
        FormsAuthentication.RedirectFromLoginPage(uname, false);
    }
    else
    {
        ClientScript.RegisterStartupScript(this.GetType(),
            "loginerr", "alert('用户名或密码不正确，请重新填写');", true);
    }
}
```

上面的代码中，我们使用了 FormsAuthentication 类的 Authenticate 方法，让 ASP.NET 页面查找存储在 web.config 文件中的证书，以进行验证。Authenticate 方法带有两个参数——要检查的用户名和密码。如果找到了证书就调用 RedirectFrom-LoginPage 方法。

在这种方式中，我们把用户的密码以明文的方式存储在 web.config 文件中，显然是不太合适的。而应该使用散列功能，防止终端用户的密码被窃取。为此，我们应在配置文件中保存散列的密码。修改的代码如下：

```
<forms loginUrl="Login.aspx" defaultUrl="Index.aspx">
   <credentials passwordFormat="MD5">
       <user name="jh" password="373633EC8AF28E5AFAF6E5F4FD87469B" />
   </credentials>
</forms>
```

使用这种构建方式，唯一要修改的是 web.config 文件。不必修改登录页面或应用程序的其他页面。要存储散列的密码，应使用 FormsAuthentication 类的 HashPasswordFor-StoringInConfigFile 方法。

通过上面的学习，我们知道Forms验证关键是基于Cookie来实现用户身份验证。它是目前使用最为广泛的身份验证方式，这主要是因为该验证方式具有很强的灵活性和适应性，能够满足多种应用需求。通常情况下，用户利用Forms验证访问受保护资源，包括以下4个步骤：

(1) 假设用户请求受保护的页面Default.aspx。

(2) HTTP模块调用Forms验证服务截取来自用户的请求，并检查其中是否包含用户凭据。

(3) 如果没有发出任何用户凭据，将自动转向用户登录页面Login.aspx。

(4) 原请求页面地址Default.aspx将以ReturnUrl值的形式附加在登录页面Login.aspx的URL地址后。当用户通过验证后，应用程序将根据ReturnUrl值进行页面重定向，以便访问Default.aspx。

<authorization>配置节用来对用户进行授权，在实现用户授权过程中，应该遵循以下两个应用规则：一是位于较低目录级别的配置文件中包含的规则优先于位于较高目录级别的规则。系统构造一个URL的所有规则的合并列表，其中最近(层次结构中距离最近)的规则位于列表头，来确定哪条规则优先。二是对于给定URL的一组合并的规则，系统从列表头开始，检查规则直到找到第一个匹配项为止。注意，ASP.NET的默认配置包含向所有用户授权的<allow users="*">元素。如果没有匹配的规则，则将允许请求，除非另外拒绝。如果找到匹配项并且匹配项是<deny>元素，则它将返回401状态代码。应用程序或站点可以方便地配置位于其站点或应用程序顶层的<deny users="*">元素以防止此行为。如果是<allow>匹配，则模块不执行任何操作，允许进一步处理请求。

在Example3_1中的Admin文件夹只运行管理员admin和sysadmin访问，根据“较低目录级别的配置文件中包含的规则优先于位于较高目录级别的规则”，可以在Admin文件夹下的Web.config文件中添加如下代码来实现。

```
<configuration>
  <appSettings/>
  <connectionStrings/>
```



```
  <system.web>
    <authorization>
      <allow users="admin,sysadmin"/>
      <deny users="*"/>
    </authorization>
  </system.web>
</configuration>
```

在 Example3_1 中有一个注册新用户的页面 RegNewUser.aspx，该页面应该是未注册用户也可以访问的，但<deny users="?"/>已经禁止未注册用户访问除了登录页面以外的任何其他页面，这时可以使用<location>配置节来实现。

使用<location>配置节，可以指定特定的文件或目录具有特别的访问权限。

在根目录下的 Web.config 文件中配置下面的代码即可实现 RegNewUser.aspx 页面允许任何用户访问。

```
<location path="RegNewUser.aspx">
  <system.web>
    <authorization>
      <allow users="?"/>
    </authorization>
  </system.web>
</location>
```

在这个例子中我们只有两个合法用户，所以我们可以针对每个用户进行授权。但是在实际的开发中，很显然是有成千上万个的用户，如果还是按照上面的方法，虽然可以实现需求，但是工作量是非常大的，而且也很容易出错。那么怎么办呢？这就是下一章我们将要学习的——成员资格和角色管理。

3.3　成员资格管理

在以往的开发中，我们已经习惯于自己创建成员库表、自己创建逻辑代码来实现用户身份验证功能。通常，在成员库表中包含了用户名、密码等用户凭证。当用户登录提交时，所提交的用户凭证将与成员库表中的凭证比较。如果两者匹配，则表示用户通过验证。为了完善成员资格管理，还必须实现创建和管理用户等业务。

实际上，我们会发现在 Web 应用程序中有关成员验证、管理等内容基本没有太大区别。因此，我们可以考虑将这些相对固定的内容抽象并形成独立的模型以方便我们的开发。ASP.NET 为我们实现了这个解决方案，提供了成员资格管理功能。其核心是利用内置的成员库表(SQL Server)、成员资格管理 API(Membership、MembershipUser 等)、成员资格提供程序(SqlMembershipProvider 等)，实现模块化和自动化的成员资格管理模式。

3.3.1　成员资格简介

ASP.NET 成员资格支持下列功能：

(1) 创建新用户和密码。

(2) 将成员资格信息(用户名、密码和支持数据)存储在 Microsoft SQL Server、Active Directory 或其他数据存储区。

(3) 对访问站点的用户进行身份验证。可以以编程方式验证用户，也可以使用 ASP.NET 登录控件创建一个只需很少代码或无需代码的完整身份验证系统。

(4) 管理密码，包括创建、更改和重置密码。根据用户选择的成员资格选项不同，成员资格系统还可以提供一个使用用户提供的问题和答案的自动密码重置系统。

(5) 公开经过身份验证的用户的唯一标识，用户可以在自己的应用程序中使用该标识，也可以将该标识与 ASP.NET 个性化设置和角色管理(授权)系统集成。

(6) 指定自定义成员资格提供程序，这使用户可以改为用自己的代码管理成员资格及在自定义数据存储区中维护成员资格数据。



3.3.2　Membership 类

ASP.NET 提供的成员资格管理模型如图 3-3 所示，成员资格管理 API 中包括多个类，最为重要的是 Membership 和 MembershipUser 类。利用这两个核心类，能够实现用户验证、管理、信息检索等多项功能。从图 3-3 可知，以上两个核心类在模型中起到了承上启下的作用，有效隔离了由登录控件构建的用户界面和执行数据访问的成员资格提供程序。由此可知，Membership 和 MembershipUser 类主要是在逻辑上实现与成员资格管理有关的功能。实际上，具体的与数据操作有关的内容均由提供程序完成。

在 ASP.NET 应用程序中，Membership 类用于验证用户凭据并管理用户设置。Membership 类可以独自使用，或者与 FormsAuthentication 类一起使用，以便创建一个完整的站点用户身份验证系统。

Membership 类具有以下几个主要功能：

- 创建和管理用户。
- 将成员资格信息存储在 SQL Server 或其他数据存储区中。
- 对访问站点的用户进行身份验证。可以以编程方式对用户进行身份验证，也可以使用登录控件创建一个只需很少代码或无需代码的完整身份验证系统。
- 管理密码，包括创建、更改、检索和重置密码等。可以选择配置成员资格管理功能，以要求一个密码提示问题及其答案来对忘记密码的用户的密码进行重置。



图 3-3

Membership 类的成员包括用于创建、更新和删除用户的方法，但是不包括用于管理角色和以编程方式设置用户能干什么和不能干什么的方法。对于这些方法，只能求助 Roles 类，我们稍后介绍该类。

Membership 以及后面将要学习的角色管理和个性化信息的功能类都存放在名称空间 System.Web.Security 中。

Membership 类的默认提供程序将用户信息以预定格式存储到一个 SQL Server 数据库 ASPNETDB 中，如果需要使用一个定制的数据库，则可以创建自己的提供程序。类的成员对象是了解和掌握一个类的核心。表 3-2、表 3-3 列出了 Membership 类的常用属性和方法。

表 3-2　Membership 类的属性

属　　性	描　　述
ApplicationName	获取或设置应用程序的名称
EnablePasswordReset	获得一个值，指示当前成员资格提供程序是否配置为允许用户重置其密码
EnablePasswordRetrieval	获得一个值，指示当前成员资格提供程序是否配置为允许用户检索其密码
MaxInvalidPasswordAttempts	获取锁定成员资格用户允许的无效密码或无效密码提示问题答案尝试次数
MinRequiredNonAlphanumericCharacters	获取有效密码中必须包含的最少特殊字符数
MinRequiredPasswordLength	获取密码所要求的最小长度
PasswordAttemptWindow	获取在锁定成员资格用户之前允许的最大无效密码或无效密码提示问题答案尝试次数的分钟数
RequiresQuestionAndAnswer	获取一个值，该值指示默认成员资格提供程序是否要求用户在进行密码重置和检索时回答密码提示问题
UserIsOnlineTimeWindow	指定用户在最近一次活动的日期/时间戳之后被视为联机的分钟数



表 3-3　Membership 类的方法

方　　法	描　　述
CreateUser	已重载。将新用户添加到数据存储区
DeleteUser	已重载。从数据库中删除一个用户
FindUsersByEmail	已重载。获取一个成员资格用户的集合，这些用户的电子邮件地址包含要匹配的指定电子邮件地址
FindUsersByName	已重载。获取一个成员资格用户的集合，这些用户的用户名包含要匹配的指定用户名
GeneratePassword	生成指定长度的随机密码
GetAllUsers	已重载。获取数据库中用户的集合
GetNumberOfUsersOnline	获取当前访问应用程序的用户数
GetUser	已重载。从数据源获取成员资格用户的信息
GetUserNameByEmail	获取一个用户名，其中该用户的电子邮件地址与指定的电子邮件地址匹配
UpdateUser	用指定用户的信息更新数据库
ValidateUser	验证提供的用户名和密码是有效的

3.3.3　建立成员资格支持

要创建一个基于成员资格 API 的身份验证层，首先选择成员资格提供程序和建立数据存储。在最简单的情况下，可以使用默认的预定义提供程序。

(1) 新建 ASPNETDB 数据库：在 Visual Studio 2008 命令提示符下输入 aspnet_regsql，然后根据提示就可创建 ASPNETDB 数据库。

(2) 新建 ASP.NET 网站 Example3_2，修改配置文件，添加连接字符串和 membership 配置节，代码如下：

```
<connectionStrings>
  <clear/>
  <add name="constr"
       connectionString="server=.;database=aspnetdb;uid=sa;pwd=sa;"
```

```
        providerName="System.Data.SqlClient"/>
</connectionStrings>
...
<membership defaultProvider="SqlProvider" userIsOnlineTimeWindow="15">
  <providers>
    <clear />
    <add
      name="SqlProvider"
      type="System.Web.Security.SqlMembershipProvider"
      connectionStringName="constr"
      applicationName="abcde123"
      enablePasswordRetrieval="false"
      enablePasswordReset="true"
      requiresQuestionAndAnswer="true"
      requiresUniqueEmail="true"
      passwordFormat="Hashed" />
  </providers>
</membership>
```



注意

membership 中 add 节点 connectionStringName 属性的值必须和连接字符串的名字一样，membership 节点 defaultProvider 属性值必须和 add 节点 name 属性值一样。

(3) 创建成员资格用户。在“网站”菜单上单击“ASP.NET 配置”，如图 3-4 所示。



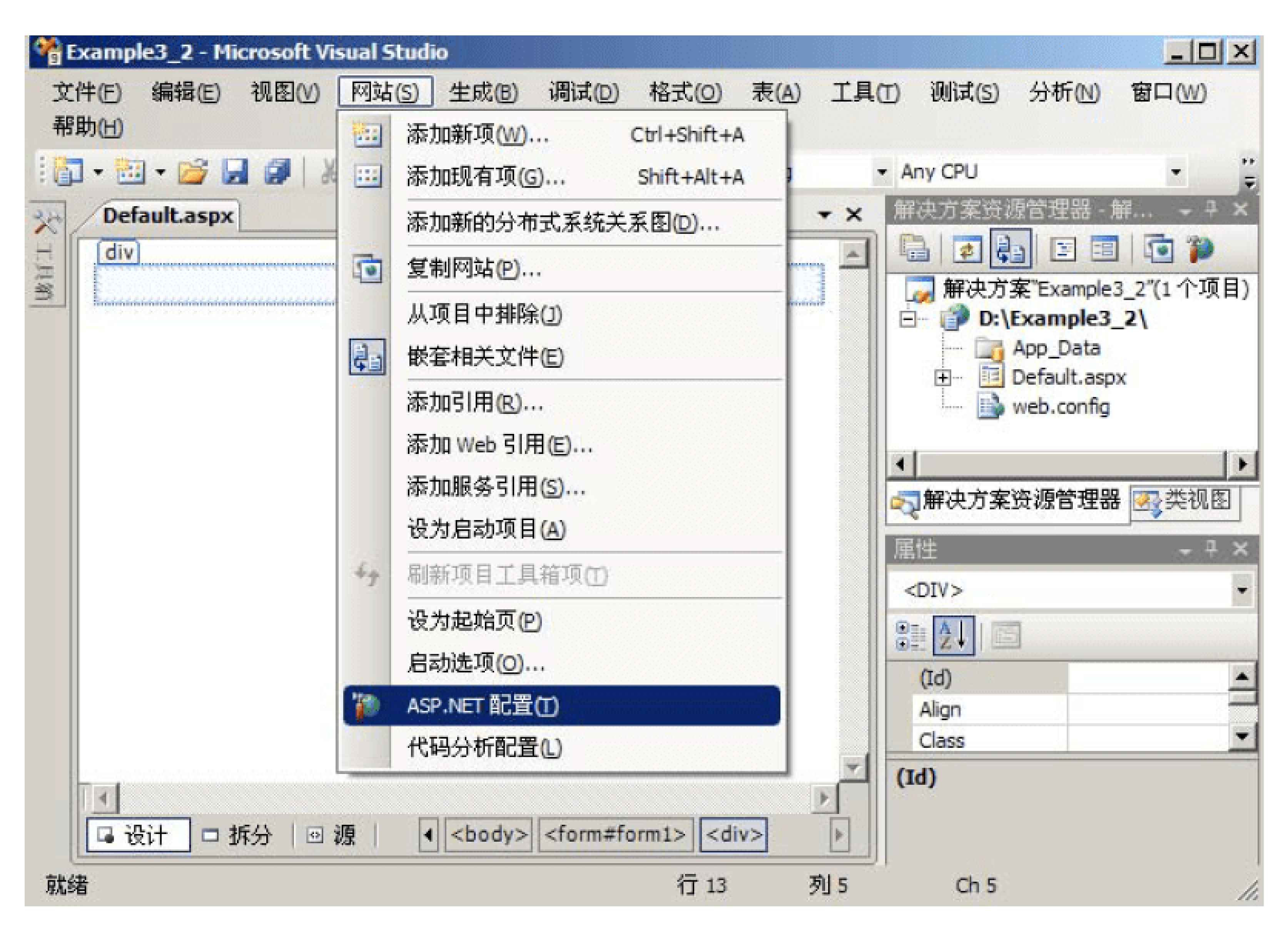


图 3-4

(4) 选择“安全”选项卡，然后单击指向“使用安全设置向导按部就班地配置安全性”的链接，再单击“下一步”按钮，如图 3-5 所示。

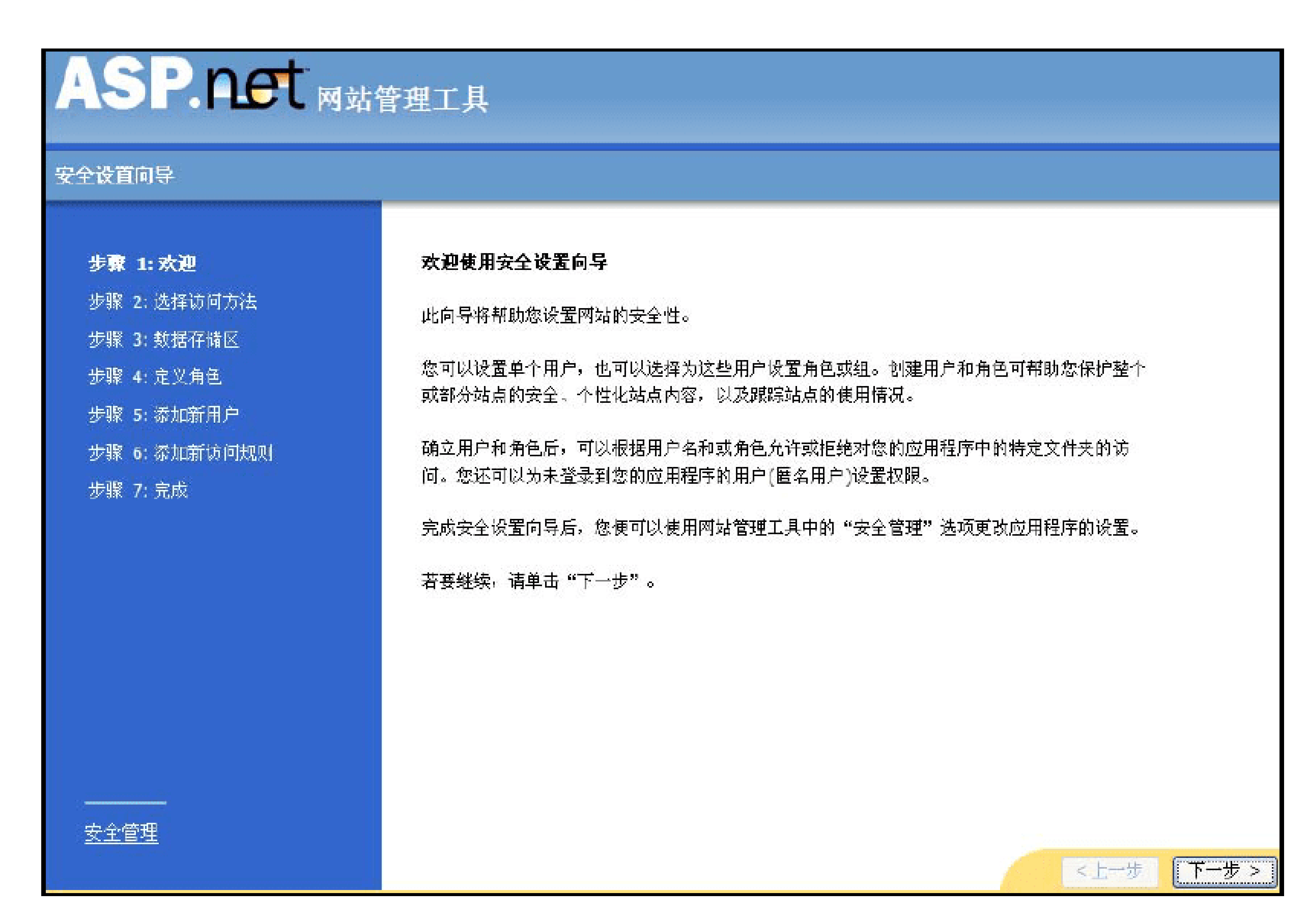


图 3-5

(5) 继续执行向导的第 2 步并选择“通过 Internet”选项。该向导显示一页，从该页中用户可以选择网站使用的身份验证方法。该选项指定应用程序将使用 Forms 身份验证，意



味着用户将使用在本演练稍后部分中创建的登录页登录应用程序，如图 3-6 所示。

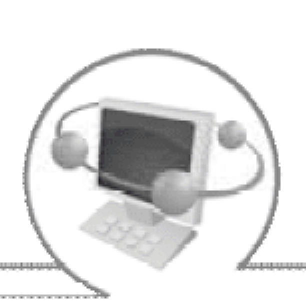


图 3-6

(6) 单击“下一步”按钮，该向导会显示一条消息，表明将使用“高级提供程序设置”存储用户信息，如图 3-7 所示。默认情况下，成员资格信息存储在网站上 App_Data 文件夹的 Microsoft SQL Server 速成版数据库文件中。



图 3-7



(7) 单击“下一步”按钮，该向导显示创建角色的选项。我们将在本演练的稍后部分中单独执行此步骤。因此，不要选择“为此网站启用角色”复选框，如图 3-8 所示。单击“下一步”按钮。



图 3-8

(8) 在该向导显示页中，用户可以创建新用户，输入定义应用程序用户的信息，如图 3-9 所示。在输入密码信息时，注意需要严格的密码(该密码包括大写和小写字母以及标点，且长度至少为八个字符，必须包括一个非字母和数字的字符)。

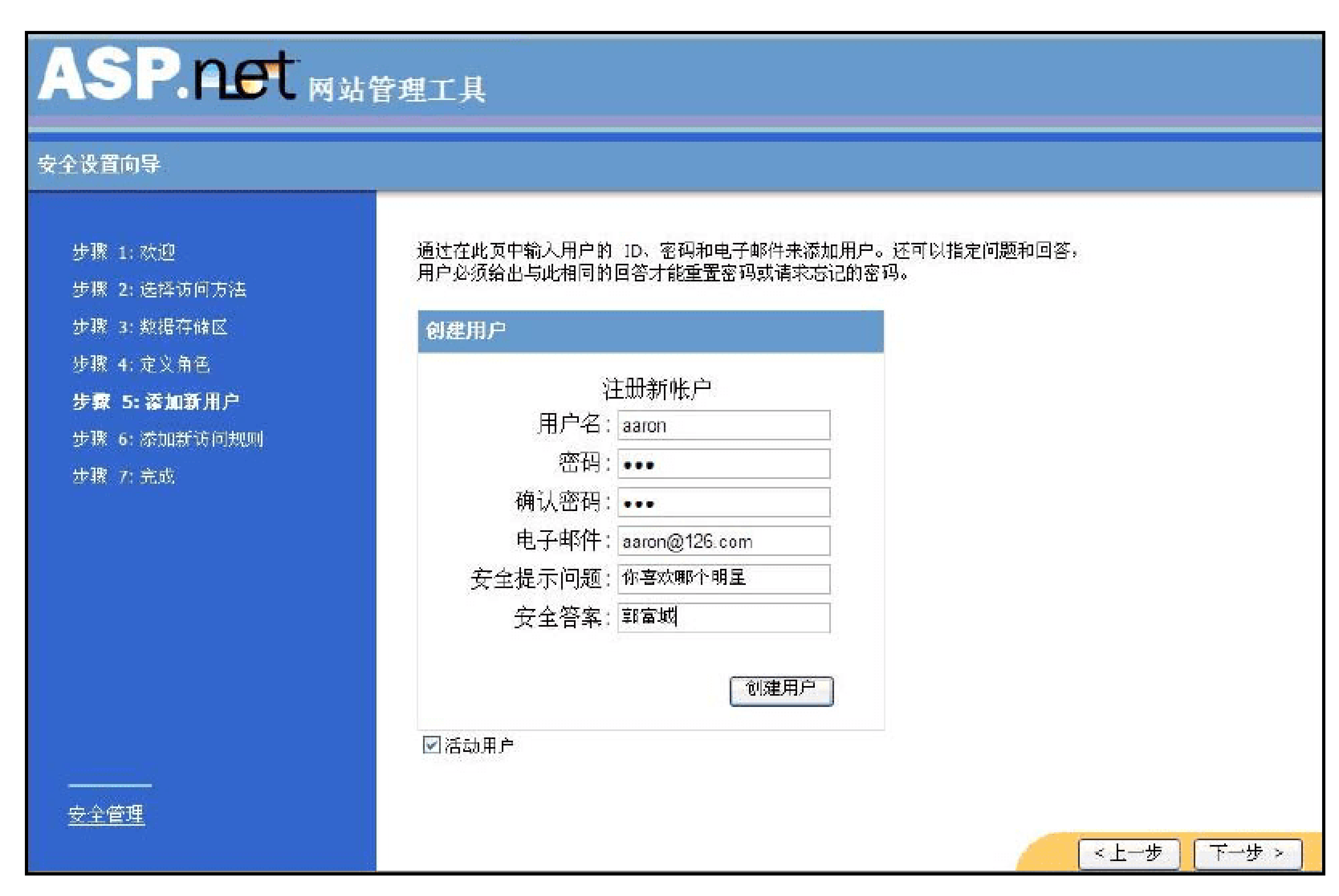


图 3-9

(9) 单击“创建用户”按钮，完成新用户创建，如图 3-10 所示。



图 3-10

3.3.4 成员资格管理实例

前面我们学习了成员资格管理功能的基本概念、Membership 类等内容，下面我们就利用这些内容，来完成一个实例，实现用户的登录验证。图 3-11 显示了示例效果图。在本例中，用户可在文本框中输入用户名和密码，然后单击“登录”按钮实现登录。当用户通过验证后，页面会自动转向到显示用户列表的页面。如果在页面 URL 中包含以 ReturnUrl 为键的 QueryString 字符串，那么页面将按照 ReturnUrl 值的设置，转向到相关页面。



图 3-11

登录页面 Login.aspx.cs 文件源代码如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    //数据绑定，显示允许错误登录的次数
```



```
    this.lblNum.Text = Membership.MaxInvalidPasswordAttempts.ToString();
}
protected void btnLogin_Click(object sender, EventArgs e)
{
    string userName = this.txtLoginName.Text;
    string pwd = this.txtLoginPwd.Text;

    //对用户进行验证，并重定向
    if (Membership.ValidateUser(userName, pwd))
    {
        if (Request.QueryString["ReturnUrl"] != null)
        {
            FormsAuthentication.RedirectFromLoginPage(userName, false);
        }
        else
        {
            FormsAuthentication.SetAuthCookie(userName, false);
            Response.Redirect("main.aspx");
        }
    }
    else
    {
        ClientScript.RegisterStartupScript(this.GetType(), "loginerr",
        "alert('无效的用户名和密码，请重新输入！ ');",true);
    }
}
```

```
protected void btnRegister_Click(object sender, EventArgs e)
{
    Response.Redirect("AddUser.aspx");
}
```

在“登录”按钮的单击事件中，首先获取提交的用户名和密码信息，然后，调用 Membership 类的验证方法 ValidateUser，并执行下述判断。如果验证通过，则进行页面重定向。一种可能是重定向到 ReturnUrl 设置的页面地址，另一种可能是重定向到主页面 Main.aspx。如果验证未能通过，则弹出提示信息。

图 3-12 显示了 AddUser.aspx 页面的效果。

图 3-12

创建用户按钮调用 Membership 的 CreateUser 方法来添加一个新用户，代码如下：

```
protected void btnAddUser_Click(object sender, EventArgs e)
{
    string name = this.txtName.Text;
    string pwd = this.txtPwd.Text;
    string email = this.txtEmail.Text;
    string question = this.txtQuestion.Text;
```

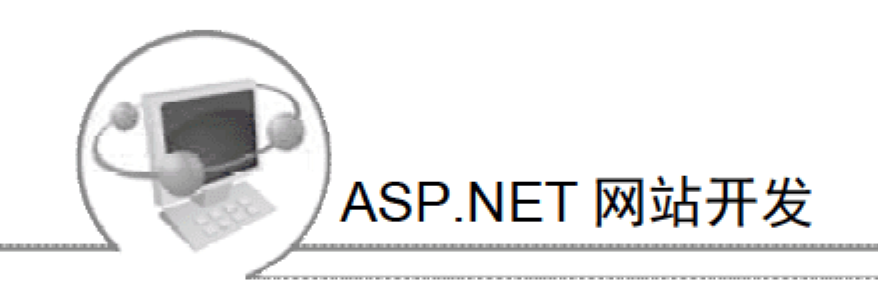


```
    string answer = this.txtAnswer.Text;
    bool active = this.chkActive.Checked;

    MembershipCreateStatus status;
    MembershipUser user = Membership.CreateUser(name, pwd, email, question,
                                  answer, active, out status);
    switch (status)
    {
        case MembershipCreateStatus.InvalidEmail:
            this.lblInfo.Text = "电子邮件不合法！";
            break;
        case MembershipCreateStatus.InvalidPassword:
            this.lblInfo.Text = "密码格式不正确！";
            break;
        case MembershipCreateStatus.Success:
            this.lblInfo.Text = "创建用户成功！";
            break;
        default:
            this.lblInfo.Text = "发生了未知错误！";
            break;
    }
}
```

3.3.5　成员资格提供程序

通过 Web.Config 文件的<membership>节，可以注册一个新的提供程序，配置节包括一个子元素<providers>，在这个元素下可以配置其他提供程序。通过<membership>节的 defaultProvider 属性，可以更改默认的提供程序。



3.4 角色管理

前面我们学习了成员资格管理功能，利用这个技术特性，我们可以轻松实现登录验证、创建和管理用户等功能。然而，应用程序却没有对用户进行访问授权，就是在用户登录之后，没有设置用户具有的访问权限。在实际的应用中，通常有两种方法解决以上问题：一种是实现单个用户权限信息的设置，第二种是定义不同的角色，把用户映射到这些角色上，然后再为这些角色授权。第一种方法，我们在上一章中已经学习过，实现较简单。但是在用户很多、权限很细的情况下，可实践性较差，不太可取。而第二种方法采用了角色控制思想，实现了用户与访问权限的逻辑分离，更符合企业的用户、组织、数据和应用特征。下面我们就来学习 ASP.NET 内置的角色管理功能。

在学习角色管理功能之前，我们首先要理解清楚角色的概念。角色是指具有相同权限的一类用户或者用户组，而不是指单个用户。角色与授权之间有着密切关系。ASP.NET 提供的角色管理将整个控制过程分成两个步骤：

(1) 访问权限与角色关联。

(2) 角色与用户关联。

通过这两个步骤的操作，实现了用户与访问权限的逻辑分离。管理人员授权时，是为角色授权，其影响的是角色中的多个用户，这大大提高了权限管理的效率和可控性。例如，假设一名用户的职位发生变化，那么只要删除为该用户分配的角色，然后设置代表新职位的角色即可。角色/权限之间的变化比角色/用户之间的变化要慢得多，并且委派用户到角色不需要复杂的技术，可以由行政人员来执行，配置权限到角色的工作比较复杂，可以由专门的技术人员来承担，但是不给他们委派用户的权限，这与现实中的情况正好一致。

实现基于角色的授权管理功能包括两个关键步骤。首先，定义所有可能的角色，并为每个用户分配角色。典型的做法是在存储用户信息的数据表中加入一列来存储每个用户的角色。然后，将每个用户与角色关联。这种方法比较繁琐，实现起来也有难度。ASP.NET 的角色管理功能，解决了上面的问题，它的实现与成员资格管理功能相同。如图 3-13 所示，显示了角色管理功能的模型。



图 3-13

从图 3-13 中可以看出，模型分为 4 个层，当执行角色管理功能时，首先必须利用登录系列控件和其他服务器控件，实现用户登录、角色管理等用户界面。然后，可调用角色管理 API 实现角色管理功能。例如，创建新角色、为用户分配角色等。角色管理 API 中包含多个类，其中最为重要的是 Roles 类。最后，角色管理 API 将调用角色管理提供程序，以实现对角色管理数据库的访问。

角色管理 API 允许我们定义角色，以及以编程的方式指定哪些用户属于哪些角色。配置角色管理、定义角色、添加用户到角色以及建立访问规则的最容易的方法是使用 ASP.NET 网站管理配置工具(WSAT)，如图 3-14 所示。



图 3-14

单击“启用角色”超链接，选择创建和管理角色，如图 3-15 和图 3-16 所示。



图 3-15



图 3-16

通过添加如下脚本到应用程序的 Web.Config 中，可以启用角色管理：



```
<roleManager enabled="true" defaultProvider="SqlProvider">
    <providers>
        <clear/>
        <add name="SqlProvider"
                type="System.Web.Security.SqlRoleProvider"
                connectionStringName="constr"
                applicationName="abcde123"/>
    </providers>
</roleManager>
```



我们可以用角色建立页面和文件夹的访问权限规则。如下<authorization>块表明只有Admin 角色可以访问 Web.Config 文件夹的所有页面：

```
<configuration>
  <system.web>
    <authorization>
      <allow roles="Admin"/>
      <deny users="*"/>
    </authorization>
  </system.web>
</configuration>
```

3.4.1　Roles 类

ASP.NET 网站管理配置工具(WSAT)提供一个可视化界面用于建立用户和角色的关系。如果需要，可以通过调用角色的各种方法，以编程的方式执行该任务。角色管理 API 中包括多个类，其中最为重要的是 Roles 类。该类分割了用户界面与执行底层数据访问的角色管理提供程序，从而为快速实现多种数据源存储的角色管理应用提供了技术支持。下面我们就来学习 Roles 类的属性和方法。

在 ASP.NET 应用程序中，Roles 类主要用于管理角色中的用户成员资格，以便实现授权检查。Roles 类具有以下几个主要功能：

- 创建和管理角色。
- 将角色信息存储在 SQL Server 或其他数据源中。
- 获取有关角色管理配置的详细内容。

一个角色中可以包含多个用户。这样，当为某个角色授权时，实际上也就完成了对角色中多个用户的授权。一个用户可以属于多个角色。例如，如果站点是一个论坛，则用户可能具有“普通用户”和“版主”双重角色。通常，这两个角色具有不同的权限，而具有双重角色的用户将具有这两个角色权限集。以上这些功能，只需调用 Roles 类的个别方法，

就能够轻松实现。另外 Roles 类还包含多个其他属性和方法，这些成员对象是实现 Roles 类应用的主要途径。通过 Roles 类的成员对象，可轻松实现角色管理功能。

下面表 3-4、表 3-5 分别列出了 Roles 类常用的属性和方法。

表 3-4　角色类 Roles 的主要属性

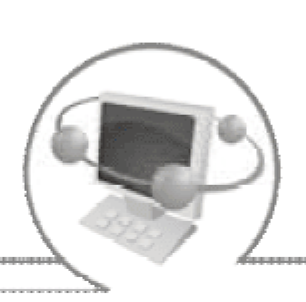
属　　性	说　　明
ApplicationName	获取或设置要存储和检索其角色信息的应用程序的名称
CacheRolesInCookie	获取一个值，该值指示当前用户的角色是否已缓存在某个 Cookie 中
CookieName	获取在其中缓存角色名称的 Cookie 的名称
CookiePath	获取缓存角色名称的 Cookie 的路径
CookieProtectionValue	获取一个指示如何保护在 Cookie 中缓存的角色名称的值
CookieSlidingExpiration	指示是否将要定期重置角色名称 Cookie 的到期日期和时间
Enabled	获取或设置用来指示是否为当前 Web 应用程序启用角色管理的值

表 3-5　角色类 Roles 的主要方法

方　　法	说　　明
AddUsersToRole	将指定的用户添加到指定的角色中
CreateRole	将新的角色添加到数据源
DeleteCookie	删除在其中缓存角色名称的 Cookie
DeleteRole	已重载。从数据源移除一个角色
FindUsersInRole	获取属于指定角色的用户列表，其中用户名包含要匹配的指定用户名
GetAllRoles	获取应用程序的所有角色列表
GetRolesForUser	已重载。获取用户所属的角色列表
GetUsersInRole	获取一个某角色所有用户的列表
IsUserInRole	已重载。获取一个指示用户是否属于指定角色的值
RemoveUserFromRole	从指定的角色中移除指定的用户

3.4.2　角色管理实例

上面我们介绍了 Roles 类的基本概念以及其常用的属性和方法，利用这些内容我们能够使用极其简单的方式，实现角色管理功能。下面我们就通过一个实例来学习一下



ASP.NET 的角色管理。该实例将完成添加角色、删除角色两个功能。示例 Example3_3 效果如图 3-17 所示。

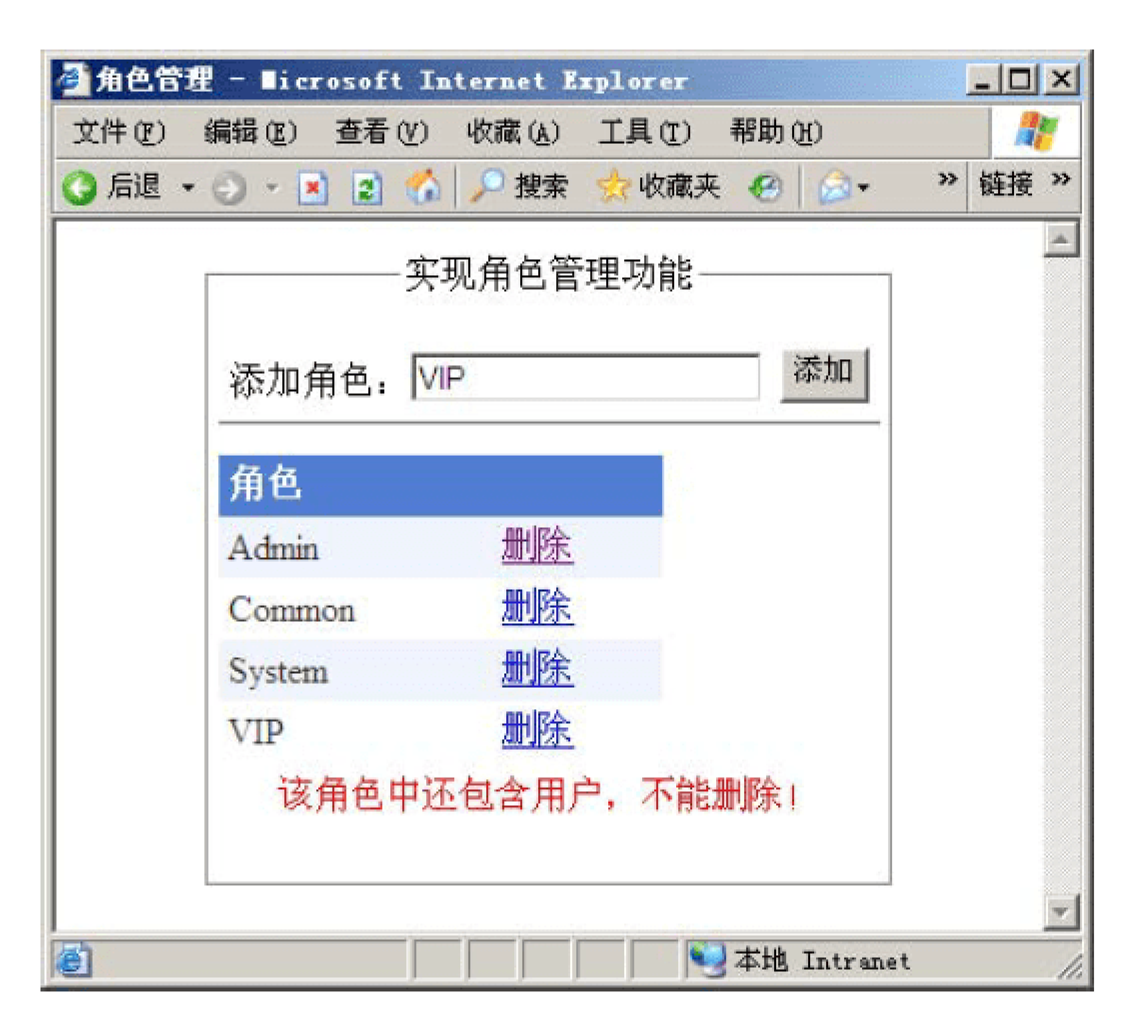


图 3-17

SimpleRoleManage.aspx 页面的设计代码如下所示。

```
<form id="form1" runat="server">
<div align="center">
<fieldset style="width: 300px">
<legend align="center">实现角色管理功能</legend>
<br />
<table border="0" cellpadding="2" cellspacing="2" width="100%">
<tr>
    <td align="center">
        添加角色：
        <asp:TextBox ID="txtRole" runat="server"></asp:TextBox>  
        <asp:Button ID="btnAdd" runat="server" Text="添加"
                    OnClick="btnAdd_Click" /><hr />
    </td>
```

```
</tr>
<tr>
    <td align="left">
        <asp:GridView ID="gvRoles" runat="server" AutoGenerateColumns="False"
            OnRowCommand="gvRoles_RowCommand" Width="196px">
            <Columns>
                <asp:TemplateField>
                    <HeaderTemplate>
                        角色
                    </HeaderTemplate>
                    <ItemTemplate>
                        <asp:Label ID="Label1" runat="server"
                       Text="<%# Container.DataItem.ToString() %>"></asp:Label>
                    </ItemTemplate>
                </asp:TemplateField>
                <asp:TemplateField>
                    <ItemTemplate>
                        <asp:LinkButton ID="lbDelete" runat="server"
                       CommandArgument="<%# Container.DataItem.ToString() %>"
                       CommandName="deleteRole">删除</asp:LinkButton>
                    </ItemTemplate>
                </asp:TemplateField>
            </Columns>
        </asp:GridView>
    </td>
</tr>
```



```
<tr>
    <td align="center">
        <asp:Label ID="lblInfo" runat="server" ForeColor="Red"></asp:Label>
    </td>
</tr>
<tr>
    <td align="right" style="height: 23px">
    </td>
</tr>
</table>
</fieldset>
</div>
</form>
```

SimpleRoleManage.aspx.cs 的完整代码如下所示。

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!this.IsPostBack)
    {
        //得到所有的角色，并绑定
        this.gvRoles.DataSource = Roles.GetAllRoles();
        this.gvRoles.DataBind();
    }
}
protected void btnAdd_Click(object sender, EventArgs e)
{
```

```
        string roleName = this.txtRole.Text;

        //如果要创建的角色不存在，则创建，否则提示
        if (!Roles.RoleExists(roleName))
        {
            Roles.CreateRole(roleName);
            this.gvRoles.DataSource = Roles.GetAllRoles();
            this.gvRoles.DataBind();
            this.lblInfo.Text = "角色已添加";
        }
        else
        {
            this.lblInfo.Text = "存在同名的角色";
        }
    }
    protected void gvRoles_RowCommand(object sender, GridViewCommandEventArgs e)
    {
        //处理角色删除
        if (e.CommandName == "deleteRole")
        {
            string roleName = e.CommandArgument.ToString();
            int userCount = Roles.GetUsersInRole(roleName).Length;

            //如果要删除的角色还包含用户，则不能删除
            if (userCount != 0)
            {
```



```
                this.lblInfo.Text = "该角色中还包含用户，不能删除！";
                return;
            }

            //调用方法删除角色，重新绑定数据
            if (Roles.DeleteRole(roleName))
            {
                this.gvRoles.DataSource = Roles.GetAllRoles();
                this.gvRoles.DataBind();
                this.lblInfo.Text = "角色删除成功！";
            }
            else
            {
                this.lblInfo.Text = "角色未能删除成功!";
            }
        }
}
```

3.5　使用用户管理控件

除了成员资格和角色管理 API 外，ASP.NET 还提供了如图 3-18 所示的服务器控件，使 Web 应用程序中与安全性相关方面的编程比以前更加容易了。这些控件与前面学习的成员资格等功能紧密结合，我们甚至无须编写一行代码就能够实现各自的用户界面和功能。同时，这些控件还内置了丰富的成员对象，大大提高了应用灵活性，另外还加强了安全性方面的控制。

新建一个网站 Example3_4 来测试用户管理控件。网站结构如图 3-19 所示。

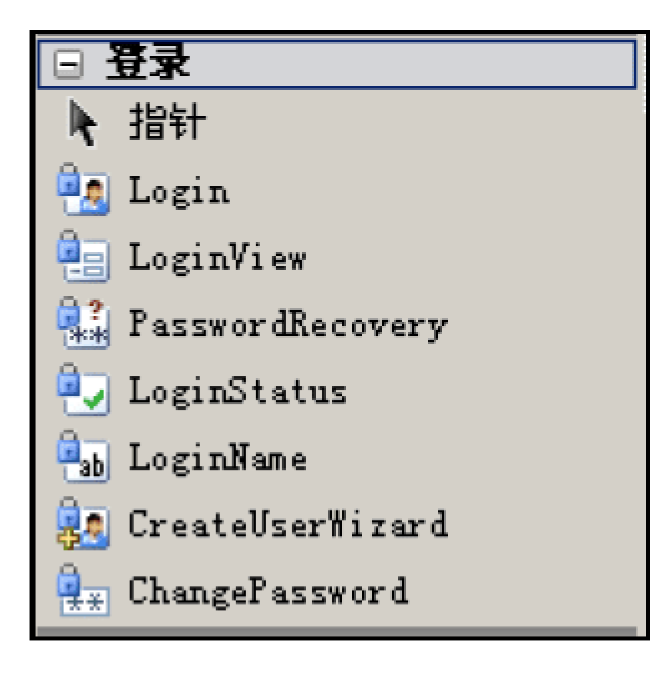


图 3-18

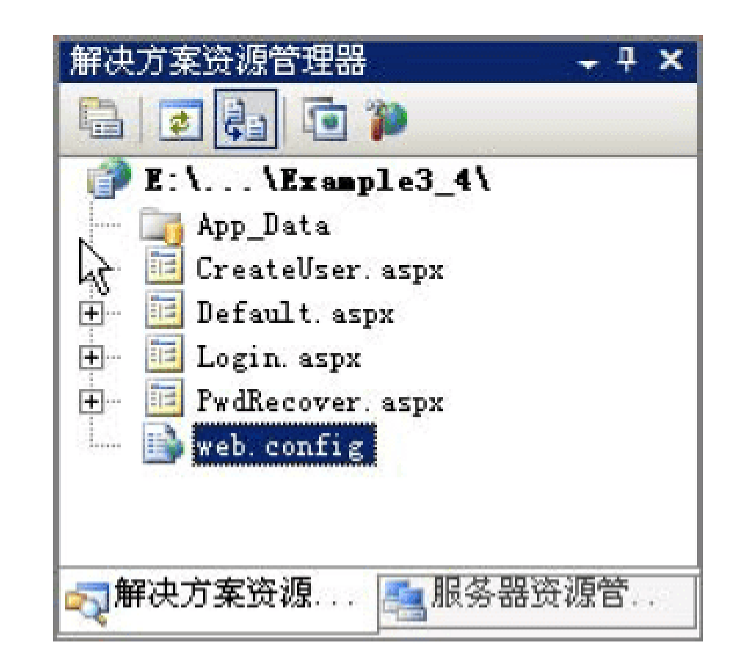


图 3-19

修改配置文件 Web.config 如下：

```
<connectionStrings>
   <clear/>
   <add name="constr"
          connectionString="server=.;database=aspnetdb;uid=sa;pwd=sa;"
          providerName="System.Data.SqlClient"/>
</connectionStrings>
...
<authentication mode="Forms">
   <forms loginUrl="Login.aspx" defaultUrl="Default.aspx"/>
</authentication>
...
<membership defaultProvider="SqlProvider" userIsOnlineTimeWindow="15">
   <providers>
      <clear />
      <add
         name="SqlProvider"
         type="System.Web.Security.SqlMembershipProvider"
         connectionStringName="constr"
```



```
            applicationName="abcde123"
            enablePasswordRetrieval="false"
            enablePasswordReset="true"
            requiresQuestionAndAnswer="true"
            requiresUniqueEmail="true"
            passwordFormat="Hashed" />
    </providers>
</membership>
...
<roleManager enabled="true" defaultProvider="SqlProvider">
    <providers>
        <clear/>
        <add name="SqlProvider"
                type="System.Web.Security.SqlRoleProvider"
                connectionStringName="constr"
                applicationName="abcde123"/>
    </providers>
</roleManager>
```

3.5.1　Login 控件

Login 控件是一个复合控件，提供了一个登录窗体的公共用户界面。图 3-20 展示了该控件的默认用户界面。要使用该控件，把它从工具箱拖放到 Login.aspx 窗体上并设置相关属性即可，或者输入如下代码：

```
<asp:Login ID="MyLogin" runat="server" CreateUserText="创建新用户"
    CreateUserUrl="~/CreateUser.aspx" DestinationPageUrl="~/Default.aspx"
    PasswordRecoveryText="找回密码" PasswordRecoveryUrl="~/PwdRecover.aspx">
```

```
</asp:Login>
```



图 3-20

Login 控件的编程接口：该控件是模块化的，并且每个组成部件都可以单独进行定制。这些部件包括“用户名”和“密码”文本框、“登录”按钮、“下次记住我”复选框。

如果不喜欢该控件的标准界面，也可以定义自己的模板：

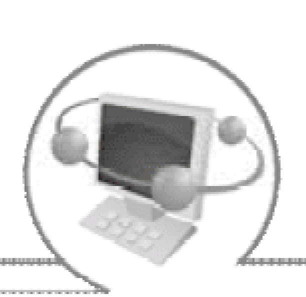
```
<asp:Login runat="Server" id="MyLogin">
   <layouttemplate>
   </layouttemplate>
</asp:Login>
```

模板中可以包括新元素，并且可以重复利用默认组件。对于后者，应当在默认模板中对这些控件使用同一 ID。在该控件的设计视图中右键单击该控件，并选择“转换为模板”，然后切换到源代码视图。

Login 控件的最常见用法是，把它作为一个单独的控件页面，建立使用窗体身份验证的登录页面。该控件完全依赖成员资格 API 来执行标准操作，诸如验证凭据、显示错误信息以及在成功登录后重定向到原始请求的页面。

3.5.2 LoginName 控件

LoginName 控件是一个极其简单而又有用的服务器控件。它的作用就像一个标签控件，



在 Web 页面上显示用户的名称。该控件从 User 内在对象获得当前用户的名称，并使用当前样式输出它。该控件在内部创建 Lable 控件的一个实例，设置相应的字体和颜色，并显示如下表达式返回的文本：

```
//User 的该属性可以返回通过窗体验证登录的用户名
string name = HttpContext.Current.User.Identity.Name;
```

3.5.3　LoginStatus 控件

LoginStatus 控件指示当前用户的身份验证状态。根据用户登录状态，它的用户界面由一个登录或者注销链接按钮组成。如果该用户充当一个匿名用户，即该用户从没有登录，则该控件显示一个链接按钮邀请该用户登录。否则，如果用户成功地通过身份验证层，则该控件显示注销链接按钮。

建立 LoginStatus 控件，LoginStatus 和 LoginName 控件结合使用，以显示当前用户的名字和一个允许用户登录和注销的按钮。与按钮变化有关的样式、文本和行动，都基于用户的身份验证状态。

```
<table width="100%" border="0">
    <tr>
        <td colspan="2">
            <asp:Label ID="Msg" runat="server" Text="Label"></asp:Label>
        </td>
    </tr>
    <tr>
        <td>
        <asp:LoginName ID="LoginName1" runat="server" FormatString="欢迎，{0}" />
        </td>
        <td align="right">
```

```
            <asp:LoginStatus ID="LoginStatus1" runat="server" LoginText="注销"/>
            </td>
        </tr>
</table>
```

图 3-21 表述了上述代码执行的结果。当用户登录成功，用 LoginName 显示登录的用户名，LoginStatus 显示为“注销”。要检查当前用户是否经过身份验证，可以使用 User 的 IsAuthenticated。

```
protected void Page_Load(object sender, EventArgs e)
{
        if (User.Identity.IsAuthenticated)
              Msg.Text = "欢迎你登录本网站";
        else
              Msg.Text = "不允许匿名登录";
}
```

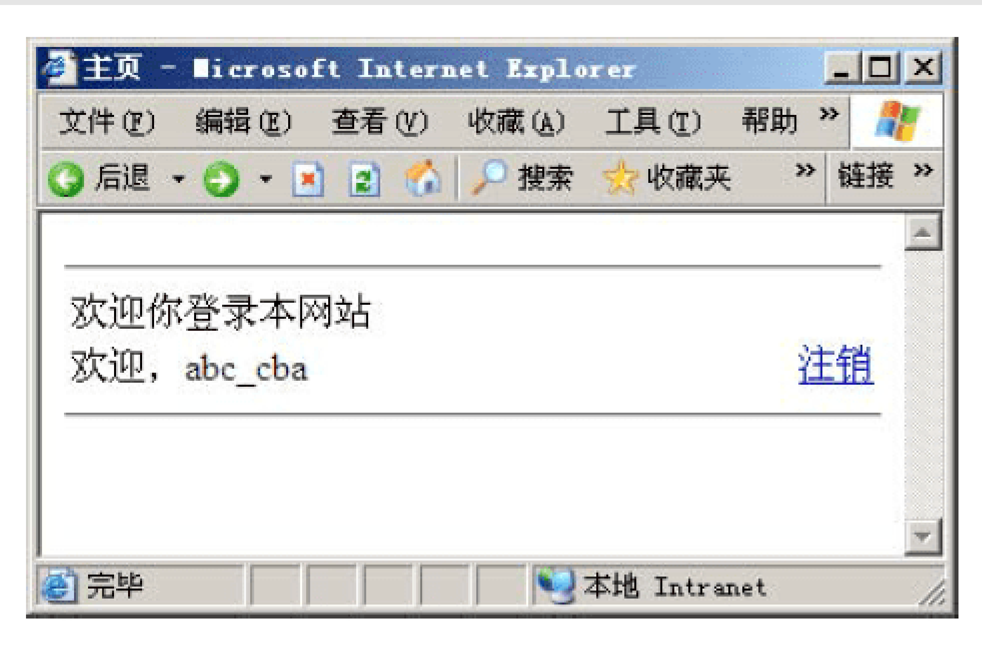


图 3-21

3.5.4 LoginView 控件

可以通过 LoginView 控件组合使用 LoginName 和 LoginStatus 控件，以显示一个定制的用户界面，同时考虑用户的身份验证状态和用户的角色。该控件基于模板，简化了匿名或者连接状态以及其所属特定角色的用户界面。



通过 LoginView 控件定义了两个不同的模板，分别向匿名用户和已登录用户显示。使用如下标记可以为页面提供一个公共的布局，并在用户登录时显示该模板。

```
<asp:LoginView ID="LoginView1" runat="server">
    <AnonymousTemplate>
            <table border="0" width="100%">
                <tr>
                    <td>
                        <h2 style="width:100%">你是匿名用户</h2>
                    </td>
                    <td align="left">
        <asp:LoginStatus ID="LoginStatus2" runat="server" LoginText="登录"/>
                    </td>
                </tr>
                </table>
        </AnonymousTemplate>
        <LoggedInTemplate>
            <table width="100%" border="0">
                <tr>
                    <td>
    <asp:LoginName ID="LoginName1" runat="server" FormatString="欢迎，{0}" />
                    </td>
                    <td align="left">
        <asp:LoginStatus ID="LoginStatus1" runat="server" LoginText="注销"/>
                     </td>
                </tr>
            </table>
```

```
        </LoggedInTemplate>
</asp:LoginView>
```

当用户匿名登录时，显示如图 3-22 所示。

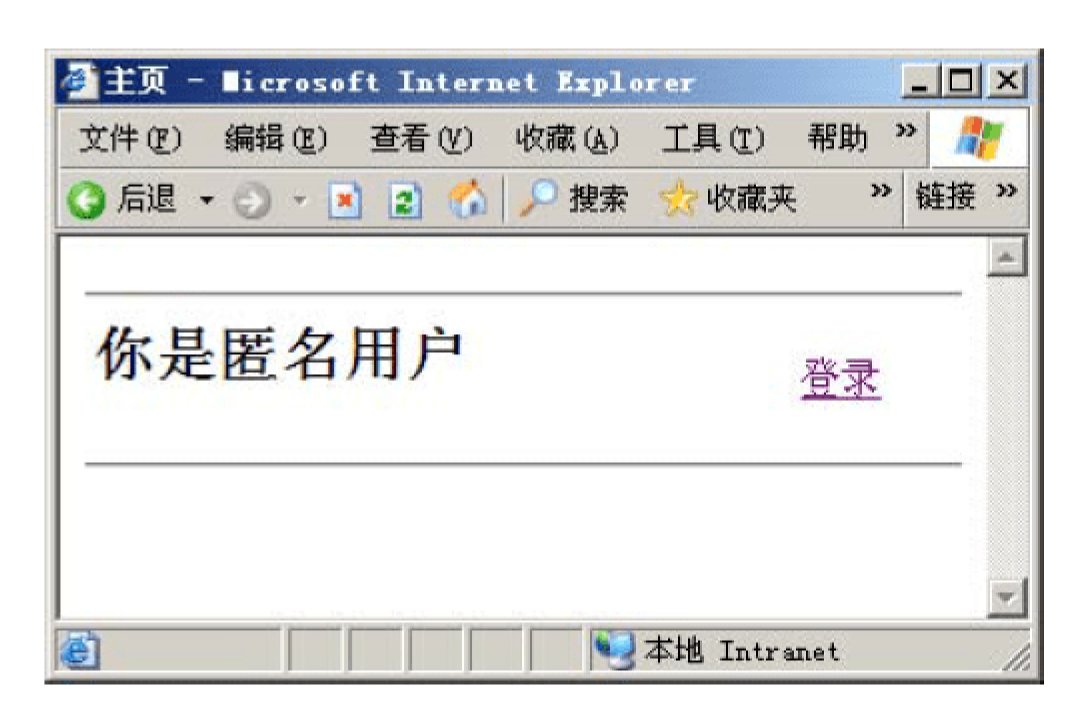


图 3-22

当用户验证登录时，显示如图 3-23 所示。



图 3-23

【小结】

- 掌握基于窗体身份的授权模式。
- 成员资格管理。
- 角色管理。
- 使用用户管理控件。



【自测题】

1. 管理注册用户的类是(　　)。

A. MemberShip　　B. MembershipUser

C. Roles　　D. FormAuthentication

2. ASP.NET 提供了多种验证方式进行身份验证，其中，能够使用 HTTP Cookies 和 HTML 表单对请求进行身份验证的是下列选项中的(　　)。

A. 窗体验证　　B. 护照验证

C. Windows 验证　　D. 定制验证

3. MemberShip 的数据提供者是下列哪一个？(　　)

A. AspNetOracleMemberShipProvider

B. AspNetSqlMemberShipProvider

C. AspNetDb2MemberShipProvider

D. AspNetMemberShipProvider

4. Roles 类的(　　)方法可以直接判断某用户是否属于某角色。

A. GetAllRoles

B. GetUsersInRole

C. IsUserInRole

D. FindUsersInRole

5. MemberShip 类的 MaxInvalidPasswordAttempts 属性其默认值是(　　)。

A. 3　　B. 5

C. 8　　D. 10

【上机部分】

上机目标

- 使用窗体身份验证技术实现登录
- 利用 Membership 类进行成员资格管理
- 利用 Roles 类进行角色管理

上机练习

◆ 第一阶段 ◆

练习：使用窗体身份验证技术实现登录。

【问题描述】

用户登录某管理系统，要求输入用户名和密码，并可选择是否在本地保存密码，以便下次在本地登录不用再重复输入用户名和密码。当登录成功自动将用户导航到欢迎页面，否则将用户导航到错误页面。

【问题分析】

- 窗体身份验证的重点是配置 Web.Config 文件，通过验证和授权配置用户的访问权限。
- 基于身份验证的页面导航技术和客户端 Cookie 过期技术的实现。

【参考步骤】

(1) 打开 Visual Studio 2008，新建一个 ASP.NET 网站。

(2) 在解决方案资源管理器中添加新项，将 Web.Config 文件添加到本项目中。

(3) 声明两个用户(用户名:LXM 密码:123 用户名:WK 密码:234)具有访问本站点的身份。



(4) 对用户密码进行 MD5 或加密。使用 FormsAuthorization 类的 HashPasswordForStoringInConfigFile 方法可以对输入的字符串进行 MD5 加密。在网站中添加一个 Public 文件夹，在 Public 文件夹下创建一个 PassWord.aspx 页面，修改配置文件，允许任何人访问该页面：

```
<location path="Public/PassWord.aspx">
   <system.web>
      <authorization>
         <allow users="?"/>
      </authorization>
   </system.web>
</location>
```

(5) 给加密按钮添加如下 Click 事件代码：

```
public partial class Public_PassWord : System.Web.UI.Page
{
    protected void btnMD5_Click(object sender, EventArgs e)
    {
           //HashPasswordForStoringInConfigFile 的第一个参数：要输入的加密字符串；
           //第二个参数：加密算法
           string strPass =
                     FormsAuthentication.HashPasswordForStoringInConfigFile(
                              txtBefore.Text, "md5");
           txtEnd.Text = strPass;
    }
}
```

运行结果如图 3-24 所示。

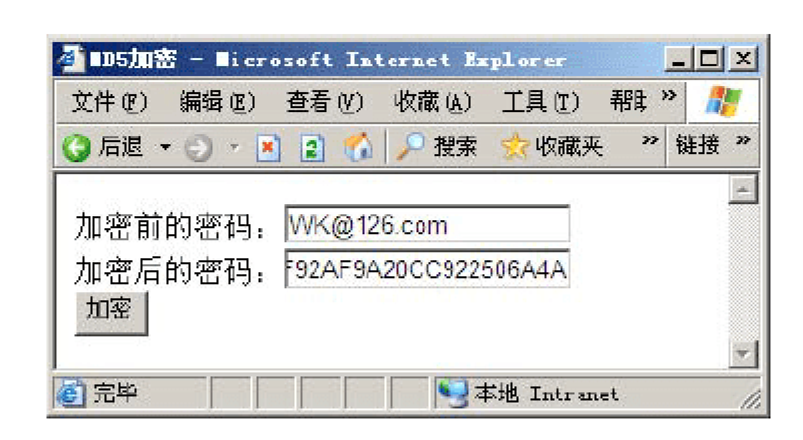


图 3-24

(6) 配置 authentication 和 authorization 配置节。授权 LXM 和 WK 用户访问本网站，禁止匿名用户访问。

```
<authentication mode="Forms">
<forms loginUrl="Login.aspx" defaultUrl="Welcome.aspx" timeout="30">
        <credentials passwordFormat="MD5">
          <user name="WK" password="7C7059490AF4F92AF9A20CC922506A4A"/>
          <user name="LXM" password="C703372355D26BC2EA766C53955F2DE2"/>
        </credentials>
  </forms>
</authentication>
<authorization>
  <allow users="WK,LXM"/>
  <deny users="?"/>
</authorization>
```

(7) 向项目添加 Login.aspx 页面，页面布局如图 3-25 所示。

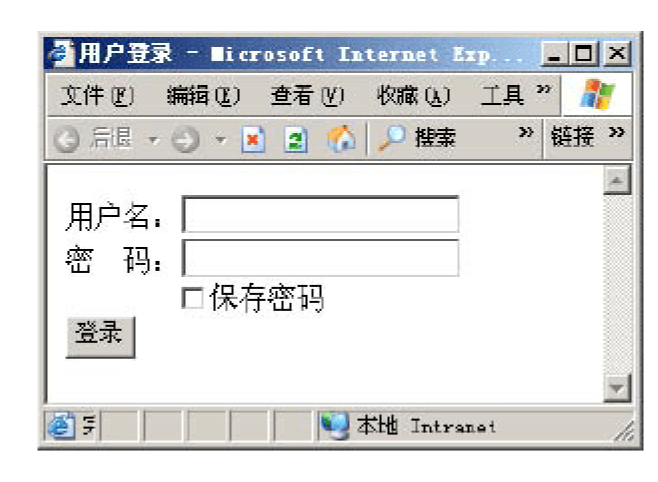


图 3-25



表 3-6 给出了 Login.aspx 主要控件的属性。

表 3-6　Login.aspx 主要控件属性

控　件	属　性	值
TextBox	ID	txtID
TextBox	ID	txtPass
CheckBox	ID	CheckPass
Button	Text	确定

(8) 在 Login.aspx.cs 页面添加以下代码片段:

```
public partial class Login : System.Web.UI.Page
{
    protected void btnLogin_Click(object sender, EventArgs e)
    {
        //是否永久保存验证 Cookie
        bool flag = CheckPass.Checked ? true : false;
        //验证方法
        if (FormsAuthentication.Authenticate(txtID.Text, txtPass.Text))
        {
            FormsAuthentication.RedirectFromLoginPage(txtID.Text, flag);
        }
    }
}
```

(9) 创建欢迎页面 Welcome.aspx，当用户通过了身份验证，将用户导航到该页面，并显示欢迎提示信息。在 Welcome.aspx 页面中添加绑定用户名的表达式:

```
<h1>欢迎光临 <% = User.Identity.Name %></h1>
```

◆ 第二阶段 ◆

练习：利用 Roles 类，完成为指定的角色添加用户的功能。

【问题描述】

在理论课的学习中，我们利用 Roles 类完成了添加角色、删除角色的功能。在这一阶段的学习中，我们将继续利用 Roles 类完成对指定角色添加用户的功能。

【问题分析】

在实际开发中，角色管理最为常见的两种应用，一种是为角色设置用户，另一种是为用户设置角色。为角色添加用户，主体是角色，其功能包括创建/删除角色、为单个角色分配/删除多个用户等。在完成功能时，主要调用的是 Roles 类的 AddUserToRole 和 RemoveUserFromRole 等方法。

【参考步骤】

(1) 打开理论课中创建的网站 Example3_3，添加一个 Web 窗体 SetRoles.aspx。

(2) 在页面中添加 Web 控件，设计页面。页面源码如下所示:

```
<form id="form1" runat="server">
    <div>
    <fieldset style="width:300px">
    <legend>用户角色管理功能</legend>
    <table border="0" cellpadding="2" cellspacing="2" width="100%">
        <tr>
            <td align="center"><b>角色管理</b></td>
        </tr>
        <tr>
            <td align="center">添加角色:
                <asp:TextBox ID="txtRole" runat="server"></asp:TextBox>
                <asp:Button ID="btnAdd" runat="server" Text="添加"
```



```
                    onclick="btnAdd_Click" /><hr /></td>
        </tr>
        <tr>
            <td align="left">
                <asp:GridView ID="gvRoles" runat="server"
                    AutoGenerateColumns="False"
                    CellPadding="4" ForeColor="#333333" GridLines="None"
                    onrowcommand="gvRoles_RowCommand" Width="230px">
                    <FooterStyle BackColor="#507CD1" Font-Bold="True"
                     ForeColor="White" />
                    <RowStyle BackColor="#EFF3FB" />
                    <Columns>
                        <asp:TemplateField>
                            <HeaderTemplate>
                                角色
                            </HeaderTemplate>
                            <ItemTemplate>
                   <asp:Label ID="Label1" runat="server"
                    Text="<%# Container.DataItem.ToString() %>"></asp:Label>
                            </ItemTemplate>
                        </asp:TemplateField>
                        <asp:TemplateField>
                            <ItemTemplate>
                   <asp:LinkButton ID="lbEdit" runat="server"
                    CommandArgument="<%# Container.DataItem.ToString() %>"
                    CommandName="editRole">编辑</asp:LinkButton>
```

```
                    </ItemTemplate>
                </asp:TemplateField>
                <asp:TemplateField>
                    <ItemTemplate>
        <asp:LinkButton ID="lbDelete" runat="server"
         CommandArgument="<%# Container.DataItem.ToString() %>"
         CommandName="deleteRole">删除</asp:LinkButton>
                    </ItemTemplate>
                </asp:TemplateField>
            </Columns>
        </asp:GridView>
    </td>
</tr>
<asp:Panel ID="plUsers" runat="server" Visible="false">
    <tr>
        <td align="left">
            <hr />
<asp:GridView ID="gvUsers" runat="server" AutoGenerateColumns="False"
 BorderWidth="1px" Width="230px">
    <Columns>
        <asp:BoundField DataField="UserName" HeaderText="用户名" />
        <asp:TemplateField>
            <HeaderTemplate>
                是否属于角色
            </HeaderTemplate>
            <ItemTemplate>
```



```
                    <asp:CheckBox ID="cbCheck" runat="server"
                     AutoPostBack="True" ToolTip='<%# Bind("UserName") %>'
                     oncheckedchanged="cbCheck_CheckedChanged" />
                </ItemTemplate>
            </asp:TemplateField>
        </Columns>
    </asp:GridView>
                </td>
            </tr>
        </asp:Panel>
        <tr>
            <td align="center">
            <asp:Label ID="lblInfo" runat="server" ForeColor="Red"></asp:Label>
            </td>
        </tr>
        <tr>
            <td align="right" style="height:23px"></td>
        </tr>
    </table>
    </fieldset>
    </div>
</form>
```

(3) 页面后台代码如下所示:

```
protected void Page_Load(object sender, EventArgs e)
{
```

```
        if (!this.IsPostBack)
        {
            //绑定所有角色
            this.gvRoles.DataSource = Roles.GetAllRoles();
            this.gvRoles.DataBind();
            //绑定所有用户
            this.gvUsers.DataSource = Membership.GetAllUsers();
            this.gvUsers.DataBind();
        }
}
protected void btnAdd_Click(object sender, EventArgs e)
{
        string roleName = this.txtRole.Text;
        //如果角色不存在，创建角色，否则提示
        if (!Roles.RoleExists(roleName))
        {
            Roles.CreateRole(roleName);
            this.gvRoles.DataSource = Roles.GetAllRoles();
            this.gvRoles.DataBind();
            this.lblInfo.Text = "角色已添加";
        }
        else
        {
            this.lblInfo.Text = "存在同名的角色";
        }
}
```



```
protected void gvRoles_RowCommand(object sender, GridViewCommandEventArgs e)
{
    //实现角色编辑
    if (e.CommandName == "editRole")
    {
        string roleName = e.CommandArgument.ToString();
        this.gvUsers.Caption = "设置角色" + roleName + "的用户";
        for (int i = 0; i < Membership.GetAllUsers().Count; i++)
        {
            CheckBox cb = (CheckBox)gvUsers.Rows[i].FindControl("cbCheck");
            string userName = cb.ToolTip;
            cb.Checked = Roles.IsUserInRole(userName, roleName);
            cb.Attributes["role"] = roleName;
        }
        this.plUsers.Visible = true;
    }
    //实现角色删除
    if (e.CommandName == "deleteRole")
    {
        string roleName = e.CommandArgument.ToString();
        int userCount = Roles.GetUsersInRole(roleName).Length;
        //如果指定的角色还包含用户，则不能删除
        if (userCount != 0)
        {
            this.lblInfo.Text = "该角色中还包含用户，不能删除！";
```

```
                return;
            }
            if (Roles.DeleteRole(roleName))
            {
                this.gvRoles.DataSource = Roles.GetAllRoles();
                this.gvRoles.DataBind();
                this.lblInfo.Text = "角色删除成功！ ";
            }
            else
            {
                this.lblInfo.Text = "角色未能删除成功!";
            }
        }
}
protected void cbCheck_CheckedChanged(object sender, EventArgs e)
{
        try
        {
            //获取角色、用户信息
            CheckBox cb = (CheckBox)sender;
            string userName = cb.ToolTip;
            string roleName = cb.Attributes["role"];

            //如果选中为角色添加用户，如果未选中则删除角色与用户的关系
            if (!cb.Checked)
```



```
            {
                Roles.RemoveUserFromRole(userName, roleName);
            }
            else
            {
                Roles.AddUserToRole(userName, roleName);
            }
        }
        catch (Exception ex)
        {
            this.lblInfo.Text = ex.Message;
        }
}
```

(4) 保存调试，运行效果如图 3-26 所示。



图 3-26

【课后作业】

在理论课 Example3_2 中只完成了登录和添加用户的功能，现要求在 UserList.aspx 页面中完成显示用户列表的功能，在 EditUser.aspx 中完成修改用户信息(包括重置用户所属角色)的功能。参考界面如图 3-27 和图 3-28 所示。



图 3-27

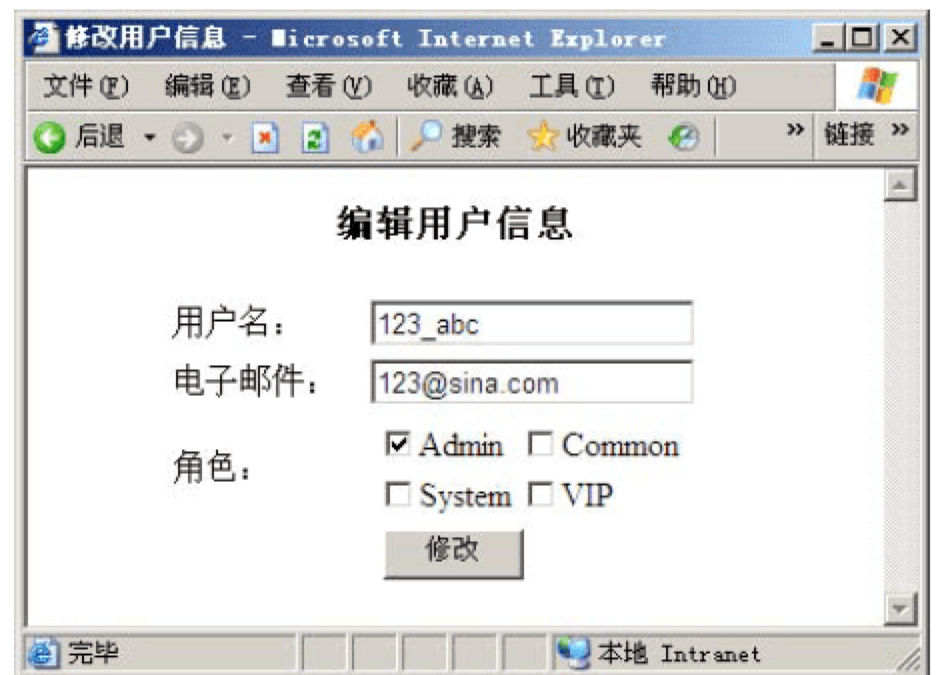


图 3-28

第4章

个性化用户配置



课程目标

- ▶ 个性化简介
- ▶ 个性化用户配置概述

简　介

随着计算机和网络技术的迅速发展，Web 站点如雨后春笋般出现。大多数 Web 站点都有一个共同的特点，即用户是以匿名方式访问站点，所有的用户看到的页面内容都是一样的。然而，对于应用要求越来越高的用户，这是无法满足其个性化需求的。问题的焦点在于用户希望实现诸如内容定制、布局调整等功能，Web 站点却很难提供这些个性化服务。因此，这样直接影响到站点的影响力和吸引力。实际上，作为 Web 站点一方而言，非常希望为用户提供个性化功能。但是，由于技术难度、时间进度、开发成本等多方面因素影响，个性化服务的梦想一直以来如空中楼阁般难以实现。

ASP.NET 技术为我们提供了一个个性化服务解决技术框架。该框架主要包括三个核心功能：个性化用户配置、Web 部件、成员和角色管理。

4.1　个性化用户配置概述

在 ASP.NET 1.x 时代，实现处理用户配置信息主要有两种方法：一种是使用数据库存储信息，还有一种是使用 Session、Application 对象。虽然这种方法简单易用，但是容易发生丢失数据的情况。例如，当重新启动服务器后，先前保存在服务器 Session 和 Application 对象中的临时用户信息就可能丢失。为了解决以上问题，从 ASP.NET 2.0 开始，新增了个性化用户配置功能。其提供的个性化用户配置功能可以实现将配置信息与某个用户关联，并采取持久化方式存储到 ASPNETDB 数据库中，这些工作都是自动完成的。为了保证个性化用户配置文件的简单性、实用性，还提供了可从应用程序中任何位置访问的多种强类型 API，以方便存储、显示和管理用户配置信息。

使用个性化用户配置功能主要包括以下两个核心步骤：首先，配置应用程序，以便启用和定义要为用户存储和跟踪的配置信息。这些工作可以在 Web.Config 文件的<profile>配置节中轻松地完成。然后，使用与用户配置功能有关的强类型 API 实现对用户配置信息的存储、访问和管理。



4.2　<profile>配置节

使用个性化用户配置功能的第一个步骤是对应用程序 Web.Config 文件进行配置，以启动和定义为用户存储和跟踪的配置信息。它们都保存在<profile>配置节中。设置<profile>配置节时，经常对其中的三部分进行配置：一是<profile>自身属性设置；二是<profile>配置节的子节<properties>属性设置；三是<profile>配置节的子节点<providers>属性设置。如表 4-1 所示是<profile>配置节属性。

表 4-1　<profile>配置节属性列表

属　性	说　明
enabled	可选的 Boolean 属性。指定是否启用 ASP.NET 用户配置文件。如果为 true，则启用 ASP.NET 用户配置文件。默认值为 true
defaultProvider	可选的 String 属性。指定默认配置文件提供程序的名称。默认值为 AspNetSqlProfileProvider
inherits	可选的 String 属性。包含从 ProfileBase 抽象类派生的自定义类型的类型引用。ASP.NET 动态地生成一个从该类型继承的 ProfileCommon 类，并将该类放在当前 HttpContext 的 Profile 属性中
automaticSaveEnabled	可选的 Boolean 属性。指定用户配置文件是否在 ASP.NET 页执行结束时自动保存。如果为 true，则用户配置文件在 ASP.NET 页执行结束时自动保存。只有在 ProfileModule 对象检测到某一用户配置文件已修改的情况下，该模块才保存该配置文件。也就是在 IsDirty 属性为 true 的情况下。有关更多信息，请参见 ASP.NET 配置文件属性。默认值为 true

表 4-2 所示为<properties>子配置节<add>的属性列表，用于添加用户配置属性。

表 4-2　<properties>子配置节<add>的属性

属　性	说　明
name	必选的 String 属性。指定属性名。该值用作自动生成的配置文件类的属性的名称，并用作该属性在 Properties 集合中的索引值。该属性的名称不能包含句点(.)
type	可选的 String 属性。指定属性类型。默认值为 String

（续表）

属　　性	说　　明
provider	可选的 String 属性。指定用于存储和检索属性值的配置文件提供程序。provider 属性的值是 providers 元素中指定的某个配置文件提供程序的名称
serializeAs	可选的 SettingsSerializeAs 属性。指定数据存储区中属性值的序列化格式。默认序列化格式视具体的提供程序而定，实际所使用的序列化由提供程序确定。对于 SQL 提供程序，则为 String 序列化
allowAnonymous	可选的 Boolean 属性。指定在应用程序用户是匿名用户的情况下是否可以获取或设置属性。如果设置为 true，则在应用程序用户是匿名用户的情况下可以获取或设置属性。默认值为 false
defaultValue	如果数据存储区中没有 Profile 属性的值，则按如下所示指定默认值：如果使用 XML 序列化对属性(property)类型进行了序列化处理，则此属性(Attribute)可以设置为表示属性(Property)类型的序列化实例的 XML 字符串。如果使用二进制序列化对属性(Property)类型进行了序列化处理，则此属性(Attribute)可以设置为表示属性(Property)类型的序列化实例的 Base-64 编码字符串。如果属性为引用类型，则可以使用 Stringnull 值指示 Profile 属性应为未初始化的配置文件返回 null
readOnly	可选的 Boolean 属性。指定是否只能读取而不能设置属性。如果设置为 true，则可以读取但不可以设置属性。默认值为 false
customProviderData	可选的 String 属性。指定 customProviderData 属性(Attribute)可以设置为任意字符串值，以供属性(Property)的配置文件提供程序使用。如果设置了此属性(Attribute)，则该值放置在属性(Property)的 Attributes 集合中，通过名称 CustomProviderData 进行索引

表 4-3 所示为<providers>子配置节<add>的属性列表。

表 4-3　<providers>子配置节<add>的属性

属　　性	说　　明
name	指定提供程序实例的名称。这是用于<profile>元素的 defaultProvider 属性的值，该值将提供程序实例标识为默认的配置文件提供程序。该提供程序的 name 还用于在 Providers 集合中对该提供程序进行索引
type	指定实现 ProfileProvider 抽象基类的类型



（续表）

属　性	说　明
connectionStringName	指定在<connectionStrings>元素中定义的连接字符串的名称。指定的连接字符串将由正在添加的提供程序使用
applicationName	指定数据源中存储配置文件数据的应用程序的名称。该应用程序名称使得多个 ASP.NET 应用程序能够使用同一个数据库，而不会遇到不同应用程序存在重复配置文件数据的情况。或者，通过指定相同的应用程序名称，多个 ASP.NET 应用程序可以使用相同的配置文件信息
commandTimeout	指定在向成员资格数据源发出的命令超时之前等待的时间(以秒为单位)。SQL 提供程序在创建 SqlCommand 对象时，使用该超时属性。默认情况下，ASP.NET 配置中并未设置该属性。因此，使用 ADO.NET 默认值 30 秒。如果设置了该属性，则 SQL 提供程序对向数据库发出的所有 SQL 命令使用已配置的超时值。默认值为 30(ADO.NET 默认值)
description	指定配置文件提供程序实例的说明

以上这些表格中列举的属性都保存在<add>节中，可以表示一个属性、添加一个属性组、添加一个提供程序等。由于在实现个性化用户配置过程中必须使用 Web.Config 文件中的<profile>配置节，因此掌握以上属性的含义具有重要意义。

下面的示例给出了实现个性化用户配置中 Web.Config 文件的部分代码。

```
<configuration>
    <appSettings/>
    <connectionStrings>
        <clear/>
        <add name="constr"
            connectionString="server=.;database=aspnetdb;uid=sa;pwd=sa;"
            providerName="System.Data.SqlClient"/>
</connectionStrings>
```

```
...
<system.web>
    ...
        <profile enabled="true" defaultProvider="AspNetSqlProfileiProvider">
            <!--配置 profile 提供程序-->
            <providers>
                <clear/>
                <add name="AspNetSqlProfileProvider"
                    connectionStringName="constr"
                    applicationName="abcde123"
                    type="System.Web.Profile.SqlProfileProvider"/>
            </providers>
            <!--配置 profile 成员-->
            <properties>
                <add name="lastName"/>
                <add name="firstName"/>
                <add name="phoneNumber"/>
                <add name="birthData" type="System.DateTime"/>
            </properties>
        </profile>
    </system.web>
</configuration>
```

Profile 配置和 membership 以及 roles 不一样，它由提供程序和成员两部分组成，提供程序默认名称就是 AspNetSqlProfileProvider。



4.3　个性化用户配置 API

ProfileBase 是用户配置功能中一个非常重要的类。该类能够提供对用户配置属性的非类型化访问。当启动包含用户配置功能的应用程序时，ASP.NET 将自动创建一个类型为 ProfileCommon 的新类，该类从 ProfileBase 类继承。同时，系统将根据<profile>配置节内容，在 ProfileCommon 新类中添加强类型访问器。这样，ProfileCommon 新类的强类型访问器就能够调用 ProfileBase 基类的 GetPropertyValue 和 SetPropertyValue 方法，从而实现对用户配置属性的检索和设置，可以通过 Profile 属性实现检索和设置用户配置属性信息。

上面的示例将使用用户配置 API 创建 4 个需要保存的信息。默认存储类型是字符串。注意，虽然默认类型是字符串，但是将使用 DateTime 对象来存储 birthdate。

新建一个 ASP.NET 网站 Example4_1，添加 Welcome.aspx 页面，该页面代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Welcome.aspx.cs"
Inherits="Welcome" %>


<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>欢迎</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:LoginView ID="LoginView1" runat="server">
          <LoggedInTemplate>
            欢迎：<asp:LoginName runat="server" ID="LoginName1" />
            <br />
```

```
                <asp:HyperLink runat="server" ID="hylInfo"
                     NavigateUrl="~/ProfileInfo.aspx">
                     添加简单的个性化信息</asp:HyperLink>
            </LoggedInTemplate>
        </asp:LoginView>
    </div>
    </form>
</body>
</html>
```

在 Example4_1 中添加 ProfileInfo.aspx 页面并设计代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ProfileInfo.aspx.cs"
Inherits="ProfileInfo" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>个性化信息</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
         <table>
              <tr>
                   <td>First Name: </td>
                   <td style="width: 193px">
                     <asp:TextBox ID="firstName" Runat="server" />
```



```
            </td>
        </tr>
        <tr>
            <td>Last Name: </td>
            <td style="width: 193px">
             <asp:TextBox ID="lastName" Runat="server" /></td>
        </tr>
        <tr>
            <td>Phone number: </td>
            <td style="width: 193px">
                <asp:TextBox ID="phone" Runat="server" />
            </td>
        </tr>
        <tr>
            <td>BirthDate</td>
            <td style="width: 193px">
                <asp:TextBox ID="birthDate" Runat="server" />
            </td>
        </tr>
         <tr>
            <td align="center" colspan="2">
                <asp:Button ID="btnSave" Text="Save" Runat="server"
                    OnClick="btnSave_Click" />
             </td>
        </tr>
    </table>
```

```
        </div>
    </form>
</body>
</html>
```

Save 按钮将用户输入的信息保存到数据库，代码如下：

```
protected void btnSave_Click(object sender, EventArgs e)
{
        //如果不是匿名用户
        if (Profile.IsAnonymous == false)
        {
                Profile.lastName = this.lastName.Text;
                Profile.firstName = this.firstName.Text;
                Profile.phoneNumber = this.phone.Text;
                Profile.birthData = Convert.ToDateTime(birthDate.Text);
                Response.Redirect("Welcome.aspx");
         }
}
```

上面的代码使用 Profile 赋值给 Web.Config 中定义的 4 个个性化信息变量。赋值完毕后，程序重新导航到 Welcome.aspx 页面。

在 Welcome.aspx 页面的 Page_Load 方法中添加如下代码：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Profile.IsAnonymous == false) //如果不是匿名用户
    {
```



```
            Response.Write(Profile.lastName + "<br/>" + Profile.firstName + "<br/> "
                + Profile.phoneNumber + "<br/>" + Profile.birthData.ToString());
    }
}
```

该段代码从 Profile 读取刚才赋值的 4 个个性化变量信息，并显示在页面上。下面将 Welcom.aspx 页面设为首页，运行如图 4-1 所示。

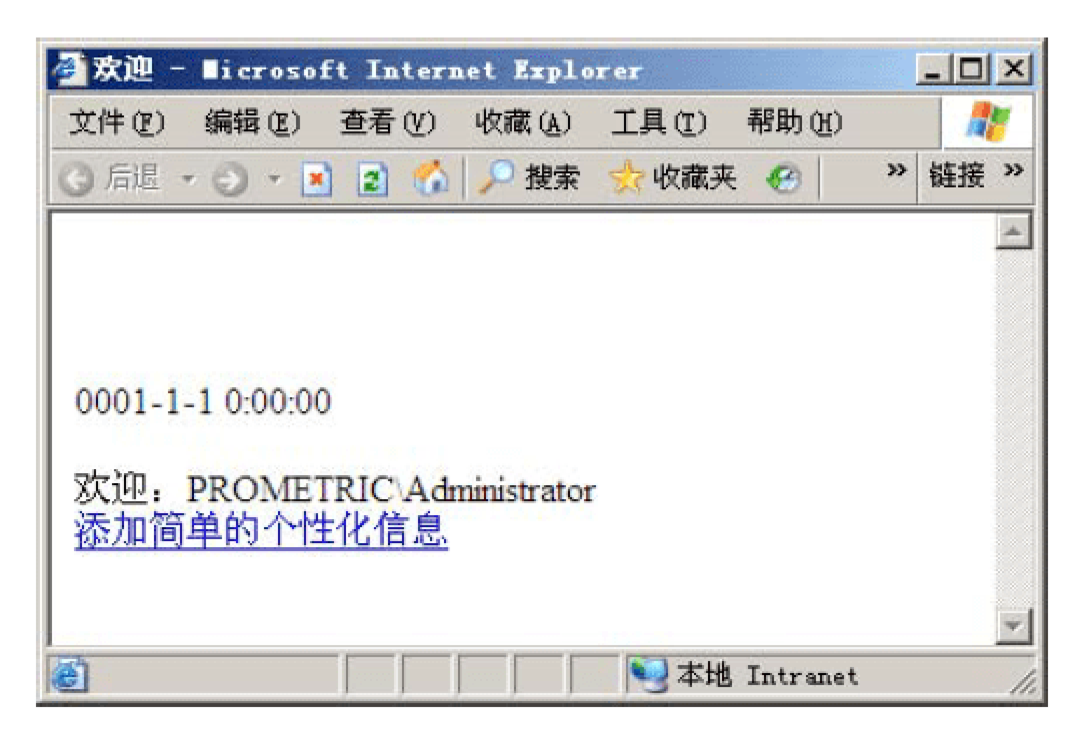


图 4-1

点击页面中的超链接，导航到 ProfileInfo.aspx 页面，在页面的文本框中输入个性化信息，如图 4-2 所示。

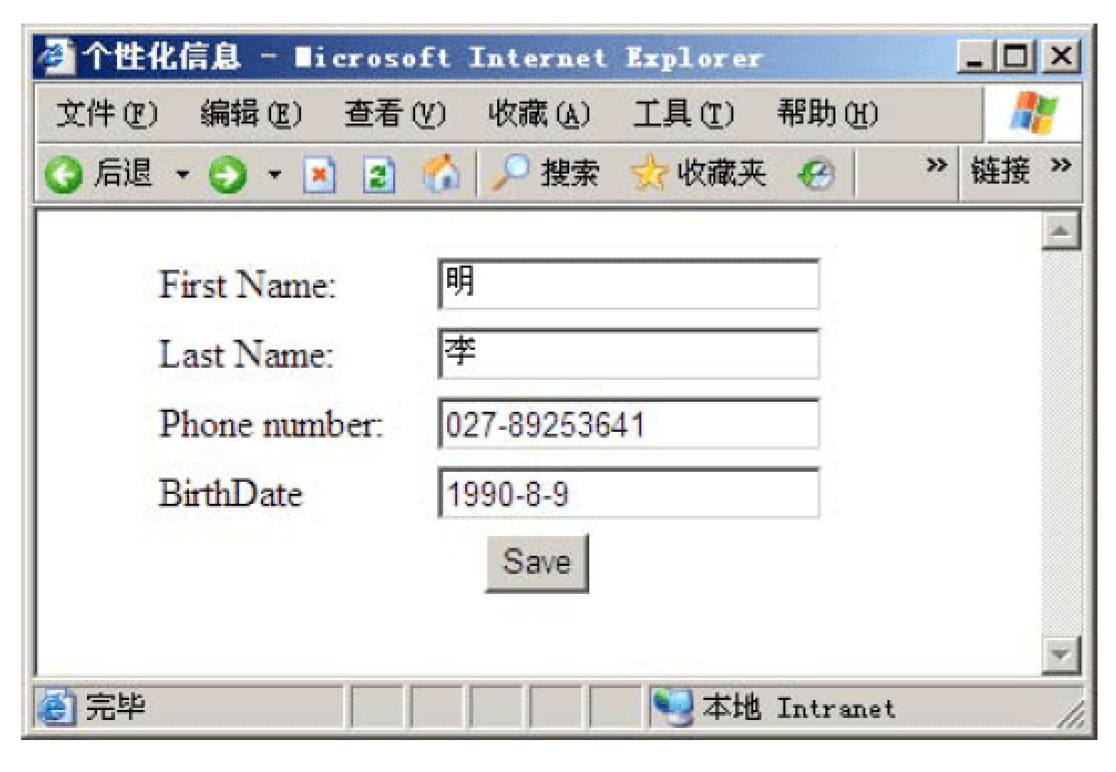


图 4-2

单击“Save”按钮保存个性化信息，重新导航到 Welcome.aspx 页面显示个性化信息，如图 4-3 所示。

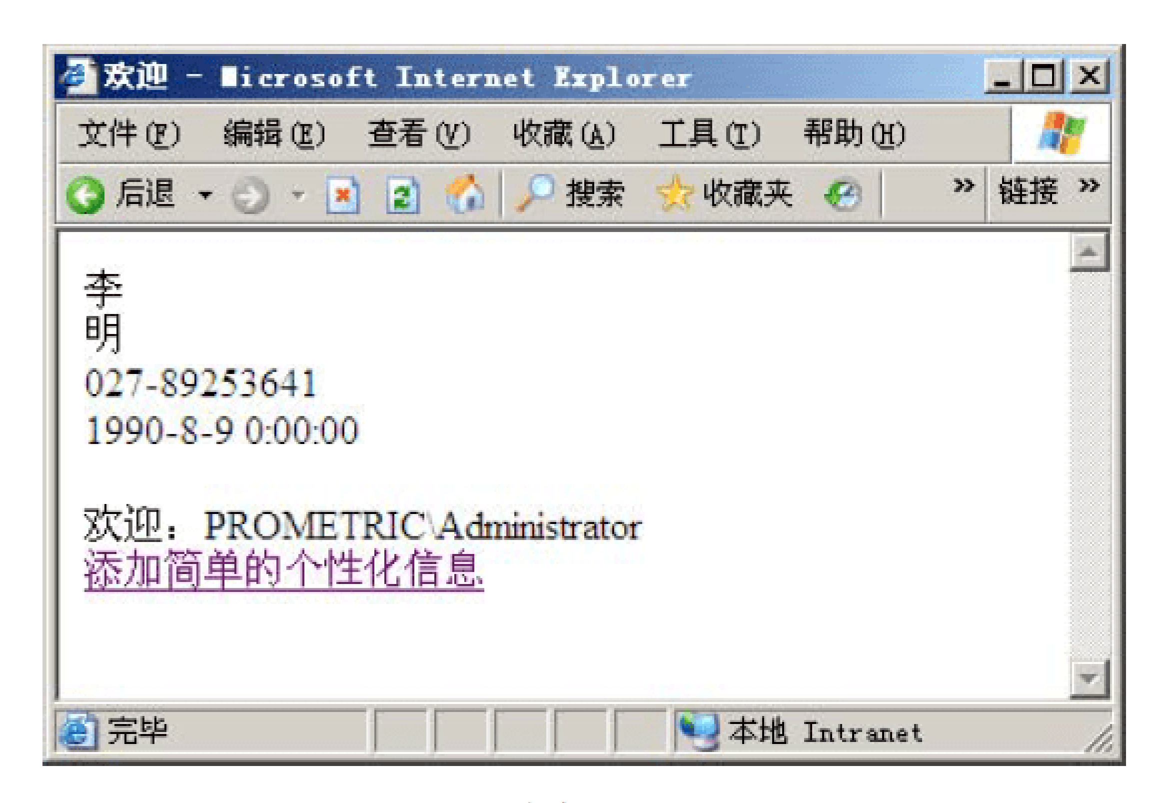


图 4-3

4.4　使用个性化配置存储复杂类型

上一节主要讲解使用个性化配置存储简单数据类型。在开发大型应用程序和商业站点的时候，经常不得不存储复杂的用户自定义数据类型和集合。

修改 Example4_1 的 Web.Config 配置文件。在个性化配置节中添加一个集合类型，代码如下所示：

```
<properties>
        <add name="lastName"/>
        <add name="firstName"/>
        <add name="phoneNumber"/>
        <add name="birthData" type="System.DateTime"/>
        <!--自定义集合类型-->
  <add name="BookDetails" type="System.Collections.Specialized.StringCollection"/>
</properties>
```

我们定义了字符串集合 BookDetails，当创建完成后，用户可以选择一本或者多本书，并且把该选择保存到用户配置中去。



重新编辑 ProfileInfo.aspx 页面，在该页面中添加一个用于显示书名的列表框，修改相应的代码如下：

```
...
<tr>
    <td   style="width: 193px">BirthDate</td>
    <td>
       <asp:TextBox ID="birthDate" Runat="server" />
    </td>
</tr>
<tr>
    <td>图书列表</td>
    <td>
      <asp:CheckBoxList ID="cblChoseBooks" runat="server">
       <asp:ListItem>C#入门经典</asp:ListItem>
       <asp:ListItem>ASP.NET 揭秘(第二版)</asp:ListItem>
       <asp:ListItem>C#高级编程</asp:ListItem>
       <asp:ListItem>Programming ASP.NET 2.0</asp:ListItem>
       <asp:ListItem>Programming C# 2.0</asp:ListItem>
      </asp:CheckBoxList>
    </td>
</tr>
 <tr>
    <td align="center" colspan="2">
       <asp:Button ID="btnSave" Text="Save" Runat="server"
```

```
            OnClick="btnSave_Click" />
    </td>
</tr>
...
```

修改 Save 按钮的事件处理程序，把选择的书名添加到用户配置中，代码如下：

```
protected void btnSave_Click(object sender, EventArgs e)
{
    //如果不是匿名用户
    if (Profile.IsAnonymous == false)
    {
        Profile.lastName = this.lastName.Text;
        Profile.firstName = this.firstName.Text;
        Profile.phoneNumber = this.phone.Text;
        Profile.birthData = Convert.ToDateTime(birthDate.Text);
        //初始化自定义集合类型
        Profile.BookDetails = new
                System.Collections.Specialized.StringCollection();
        //将选中的书名添加的集合中
        foreach (ListItem item in cblChoseBooks.Items)
        {
            if (item.Selected)
            {
                Profile.BookDetails.Add(item.Text);
            }
        }
```



```
            Response.Redirect("Welcome.aspx");
        }
}
```

在 Welcome.aspx 页面的 Page_Load 中添加一些代码，提取 ProfileInfo.aspx 页面 Profile 对象的属性值。如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    //如果不是匿名用户
    if (Profile.IsAnonymous == false)
    {
        Response.Write(Profile.lastName + "<br/>" + Profile.firstName + "<br/> "
            + Profile.phoneNumber + "<br/>"
            + Profile.birthData.ToString()+"<br/>");
        //取出选中的书名
        foreach (string strName in Profile.BookDetails)
        {
            Response.Write("书名：  " + strName + "<br/>");
        }
    }
}
```

当选择书名时，更改用户配置中的图书，如图 4-4 所示。

当单击“Save”按钮并返回到 Welcome.aspx 页面时，所存储的用户配置将显示出来，如图 4-5 所示。

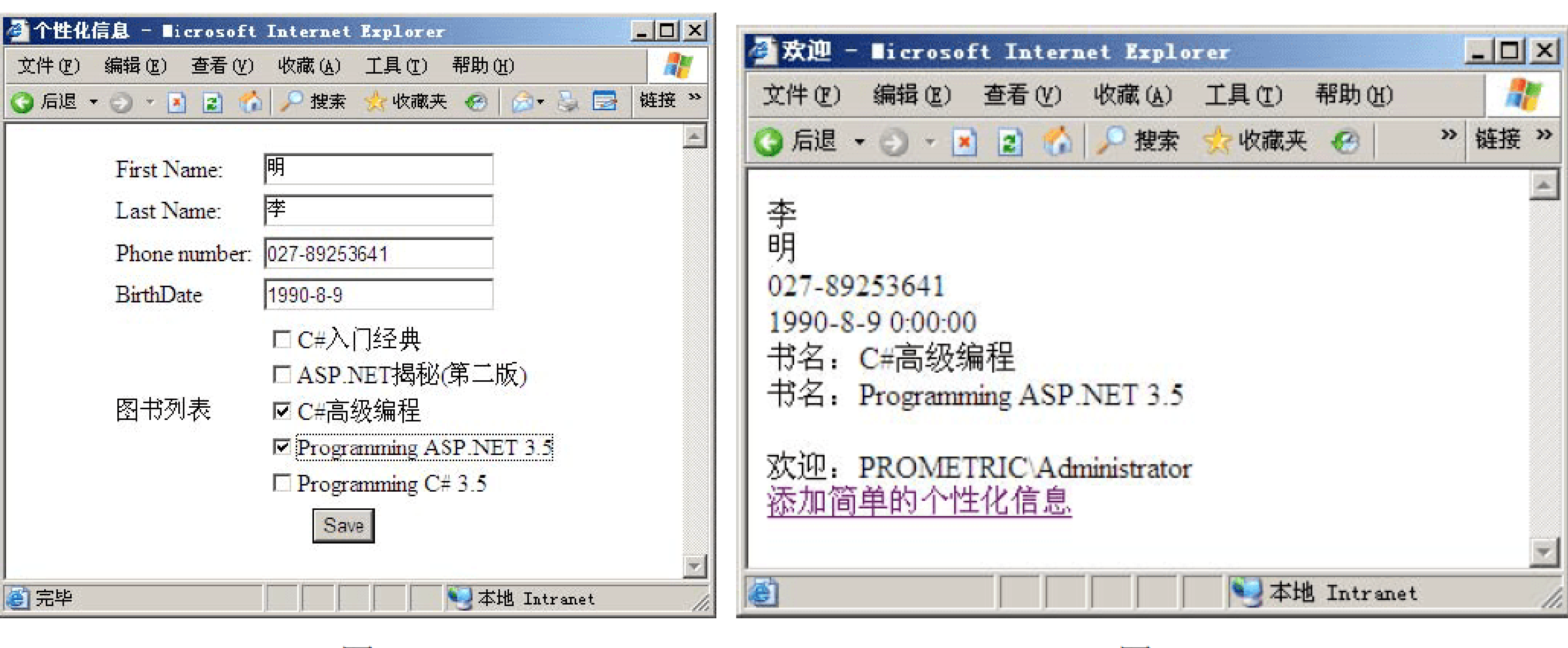


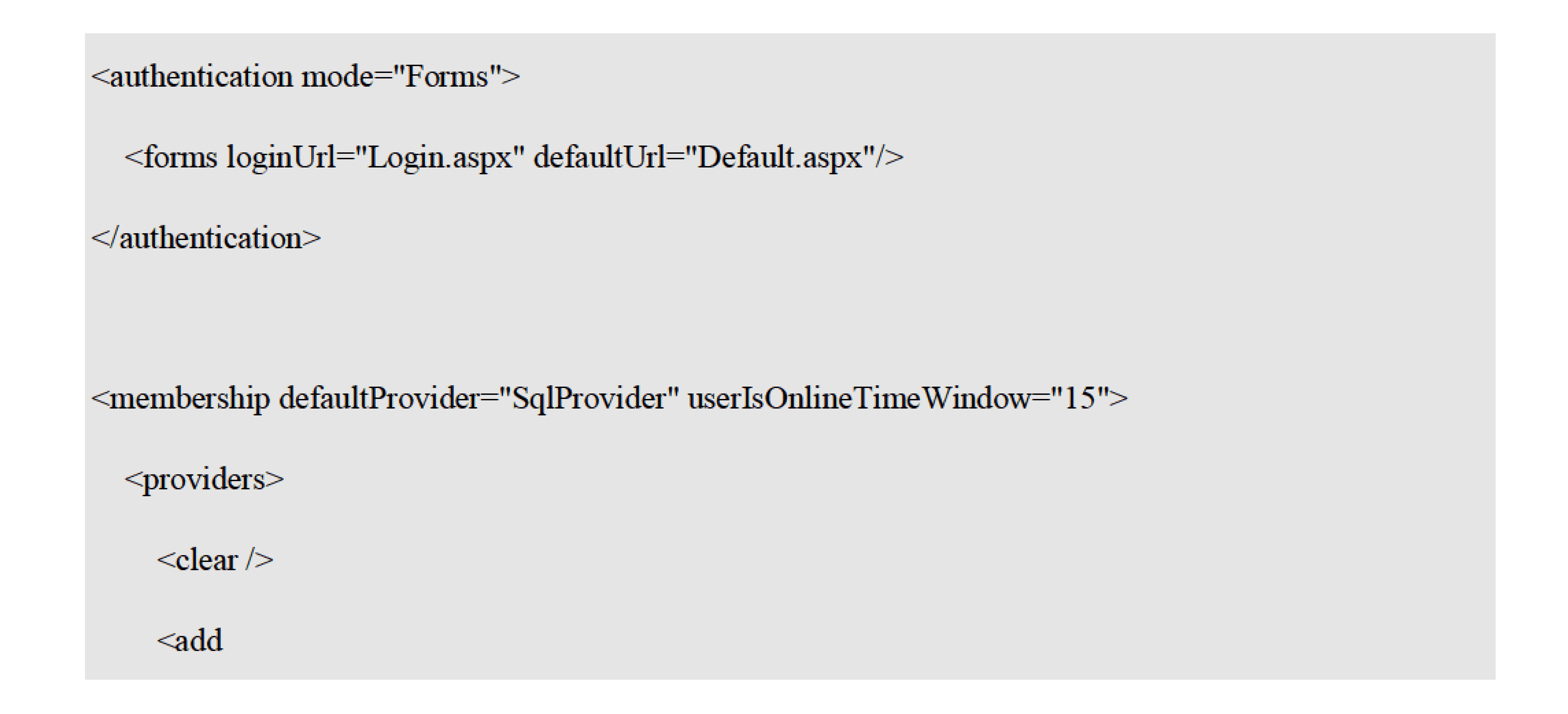
图 4-4　　　　图 4-5

4.5 匿名个性化

有这样一个场景，匿名用户通过购物车选购商品，当结账的时候显示没有登录，所以不让结账。一旦用户登录，ASP.NET 可以把匿名个性化数据与特定用户的个性化数据联系起来，并且保证用户在登录后购物车的数据不会丢失。

启用匿名个性化需要启用配置节 anonymousIdentification，并将属性 enable 设为 true。

复制 Example4_1 为 Example4_2，修改 Web.Config 配置文件，如下所示：



```
<authentication mode="Forms">
   <forms loginUrl="Login.aspx" defaultUrl="Default.aspx"/>
</authentication>

<membership defaultProvider="SqlProvider" userIsOnlineTimeWindow="15">
   <providers>
      <clear />
      <add
```



```
          name="SqlProvider"
          type="System.Web.Security.SqlMembershipProvider"
          connectionStringName="constr"
          applicationName="abcde123"
          enablePasswordRetrieval="false"
          enablePasswordReset="true"
          requiresQuestionAndAnswer="true"
          requiresUniqueEmail="true"
          passwordFormat="Hashed" />
   </providers>
</membership>

<roleManager enabled="true" defaultProvider="SqlProvider">
   <providers>
      <clear/>
      <add name="SqlProvider"
           type="System.Web.Security.SqlRoleProvider"
           connectionStringName="constr"
           applicationName="abcde123"/>
   </providers>
</roleManager>

<anonymousIdentification enabled="true"/>
<profile enabled="true" defaultProvider="AspNetSqlProfileiProvider">
   <providers>
      <clear/>
```

```
    <add name="AspNetSqlProfileiProvider"
          connectionStringName="constr"
          applicationName="abcde123"
          type="System.Web.Profile.SqlProfileProvider"/>
  </providers>
  <properties>
    <add name="lastName" allowAnonymous="true"/>
    <add name="firstName" allowAnonymous="true"/>
    <add name="phoneNumber" allowAnonymous="true"/>
    <add name="birthData" type="System.DateTime" allowAnonymous="true"/>
    <!--自定义集合类型-->
<add name="BookDetails"
       type="System.Collections.Specialized.StringCollection"
          allowAnonymous="true"/>
  </properties>
</profile>
```

将验证方式改为 Forms 验证方式，并且添加 membership 和 roleManager 配置节。要想匿名用户使用个性化配置，必须修改个性化配置节。添加 allowAnonymous 属性值为 true，否则个性化配置节将不能为匿名用户使用。除此之外，必须添加<anonymousIdentification enabled="true"/>配置节，否则编译会出错。

在 Example4_2 中添加 Login.aspx 页面，向页面拖放一个 Login 控件。

重新设计 Welcome.aspx 页。把导航到用户配置页面的超链接放在 LoginView 控件之外。这样可以在不登录的情况下看到超链接。在 LoginView 中添加 AnonymousTemplate 模板，该模板主要用于匿名登录下 LoginView 的显示样式。设计代码如下所示：

```
<form id="form1" runat="server">
```



```
<div>
    <asp:LoginStatus ID="LoginStatus1" runat="server" />
    <asp:LoginView ID="LoginView1" runat="server">
        <LoggedInTemplate>
            欢迎：<asp:LoginName ID="LoginName1" runat="server" />
            <br />
            <asp:HyperLink ID="hylInfo" runat="server"
                NavigateUrl="~/ProfileInfo.aspx">添加简单的个性化信息
            </asp:HyperLink>
        </LoggedInTemplate>
        <AnonymousTemplate>
            你还没用登录，请重新登录
        </AnonymousTemplate>
    </asp:LoginView>
<asp:HyperLink ID="hlProfile" NavigateUrl="~/ProfileInfo.aspx"
     runat="server">购书请入：
    </asp:HyperLink>
</div>
</form>
```

为了区别匿名用户和注册用户的个性化配置数据，必须修改 ProfileInfo.aspx 页。以便区别不同用户所做的操作。修改该页，让那些在 panel 中的不属于匿名用户的数据对未登录用户不可见，而让图书列表对所有用户可见(包括未登录用户)，代码如下：

```
<form id="form1" runat="server">
<div>
    <asp:Panel ID="pnlNoAnonyInfo" runat="server">
```

```
<table align="center" style="width: 311px">
    <tr>
        <td>First Name: </td>
        <td style="width: 193px">
            <asp:TextBox ID="firstName" runat="server" />
        </td>
    </tr>
    <tr>
        <td>Last Name: </td>
        <td style="width: 193px">
            <asp:TextBox ID="lastName" runat="server" />
        </td>
    </tr>
    <tr>
        <td>Phone number: </td>
        <td style="width: 193px">
            <asp:TextBox ID="phone" runat="server" />
        </td>
    </tr>
    <tr>
        <td>BirthDate： </td>
        <td style="width: 193px">
            <asp:TextBox ID="birthDate" runat="server" />
        </td>
    </tr>
</table>
```



```
    </asp:Panel>
    <table align="center" style="width: 311px">
        <tr>
            <td>图书列表</td>
            <td>
                <asp:CheckBoxList ID="cblChoseBooks" runat="server">
                    <asp:ListItem>C#入门经典</asp:ListItem>
                    <asp:ListItem>ASP.NET 揭秘(第二版)</asp:ListItem>
                    <asp:ListItem>C#高级编程</asp:ListItem>
                    <asp:ListItem>Programming ASP.NET 3.5</asp:ListItem>
                    <asp:ListItem>Programming C# 3.5</asp:ListItem>
                </asp:CheckBoxList>
            </td>
        </tr>
        <tr>
          <td align="center" colspan="2">
            <asp:Button ID="btnSave" Text="Save" runat="server"
                 OnClick="btnSave_Click"/>
          </td>
        </tr>
    </table>
</div>
</form>
```

在 ProfileInfo.aspx.cs 中的 Page_Load 事件处理程序中，添加了对匿名用户的判断，如果是匿名用户访问本页面，则让面板内的内容不可见。如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
        //判断是否为回传页面
        if (!this.IsPostBack )
        {
            //如果是匿名用户
            if (Profile.IsAnonymous == true)
            {
                //隐藏面板
                this.pnlNoAnonyInfo.Visible = false;
            }
        }
}
```

我们将 ProfileInfo.aspx.cs 的 btnSave_Click 中测试 IsAnonymous 是否为假的语句删除掉，这样无论用户是否为匿名用户，都可以对 Profile 对象进行赋值。如下所示：

```
protected void btnSave_Click(object sender, EventArgs e)
{
        Profile.lastName = this.lastName.Text;
        Profile.firstName = this.firstName.Text;
        Profile.phoneNumber = this.phone.Text;
        //如果日期文本框不为空
        if (birthDate.Text != "")
        {
            Profile.birthData = Convert.ToDateTime(birthDate.Text);
        }
```



```
        //初始化自定义集合类型
         Profile.BookDetails = new
                System.Collections.Specialized.StringCollection();
        //将选中的书名添加到集合中
        foreach (ListItem item in cblChoseBooks.Items)
        {
            if (item.Selected)
            {
                Profile.BookDetails.Add(item.Text);
            }
        }
        Response.Redirect("Welcome.aspx");
}
```

修改 Welcome.aspx 的 Page_Load 事件，不管是否是登录用户都显示全部个性化信息，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Write(Profile.lastName + "<br/>" + Profile.firstName + "<br/> "
      + Profile.phoneNumber + "<br/>" + Profile.birthData.ToString() + "<br/>");
    foreach (string strName in Profile.BookDetails)
    {
        Response.Write("书名： " + strName + "<br/>");
    }
}
```

运行网站，但不登录。单击“购书请入”超链接，选中一些书名，然后单击“Save”按钮。当返回 Welcome.aspx 页面时，虽然没有登录，却仍然显示了所选书籍的名称，如图 4-6 所示。

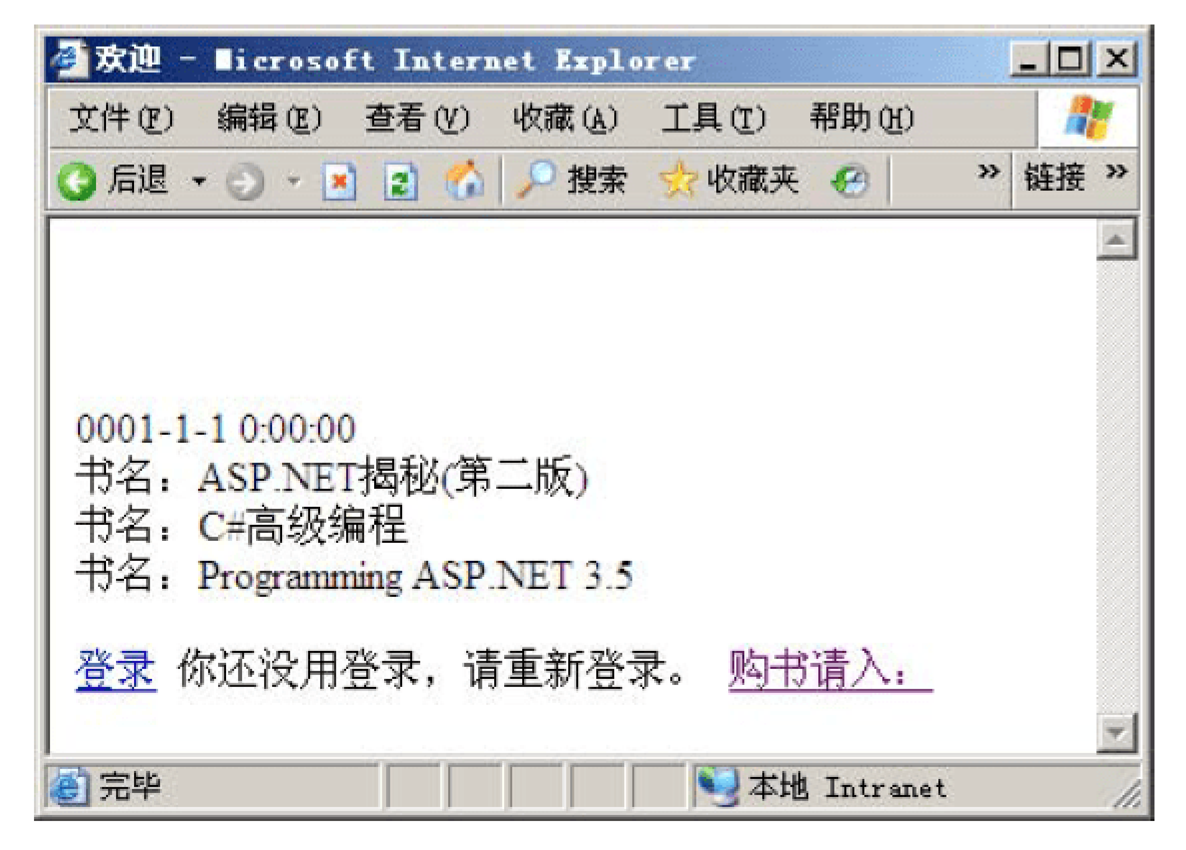


图 4-6

【小结】

- Profile 配置节可以持久化地存放用户的个性化信息。
- 个性化配置可以存储复杂的数据信息。
- 注册用户的个性化配置。
- 匿名用户的个性化配置。

【自测题】

1. 个性化是(　　)提供的技术。

A. ASP　　B. JSP

C. ASP.NET 1.x 后　　D. ASP.NET 2.0 后



2. ASP.NET 个性化启用需要配置哪个配置节？(　　)

A. Authentication　　B. SessionState

C. Profile　　D. MemberShip

3. Profile 配置节默认存放的是什么类型的数据？(　　)

A. DateTime　　B. Int32　　C. String　　D. Boolean

4. 启动 ASP.NET 匿名个性化，需要配置哪个配置节？(　　)

A. appSettings　　B. authentication

C. compilation　　D. anonymousIdentification

5. 判断是否为匿名用户要使用以下 Profile 的哪个属性？(　　)

A. IsAnonymous　　B.Properties　　C. IsSynchronized　　D. IsDirty

【上机部分】

上机目标

- 实现匿名用户的个性化用户配置
- 实现注册用户的个性化用户配置

上机练习

◆ 第一阶段 ◆

练习 1：实现匿名用户个性化用户配置。

【问题描述】

列举一个示例，实现将用户提交的姓名、国家、邮政编码和提交时间等内容，作为用户配置属性数据存放到 SQL Server 2008 中，并且在提交之后，从数据库中重新获取并显示

这些信息。

【问题分析】

实现以上示例，如果不使用个性化用户配置功能，而使用普通的数据库访问技术，是非常困难和烦琐的。难点在于应用程序必须设法为匿名用户创建唯一标识，同时还要实现繁琐的数据库访问功能。如果使用个性化用户配置功能，那么系统不仅将自动创建数据库，包括实现数据库访问，而且还会为匿名用户创建唯一标识。默认情况下，用户配置功能并不会启用对匿名用户的支持，因此，必须显示开启。当启动匿名用户支持后，用户首次访问应用程序时，ASP.NET 将为其创建一个唯一的标识。

【参考步骤】

(1) 创建一个网站，添加一个 Web.Config 文件，添加 Profile 配置节。

```
<connectionStrings>
   <clear/>
   <add name="constr"
           connectionString="server=.;database=aspnetdb;uid=sa;pwd=sa;"
           providerName="System.Data.SqlClient"/>
</connectionStrings>
...
<system.web>
<anonymousIdentification enabled="true"/>
<profile enabled="true">
   <providers>
      <clear/>
      <add name="AspNetSqlProfileProvider"
              connectionStringName="constr"
              applicationName="abchp"
              type="System.Web.Profile.SqlProfileProvider"/>
```



```
  </providers>
  <properties>
    <add name="Name" allowAnonymous="true" />
    <add name="LastSubmit" type="System.DateTime" allowAnonymous="true"/>
    <group name="Address">
      <add name="City"   allowAnonymous="true"/>
      <add name="PostalCode"   allowAnonymous="true"/>
    </group>
  </properties>
</profile>
</system.web>
```

如上代码所示，根据应用需求，<Profile>配置节添加了两个用户配置属性，它们是存储用户名称的属性 Name 和记录上次提交时间的属性 LastSubmit。同时，还创建了一个属性组 Address，其中包括两个属性：City(国家属性)和 PostalCode(邮编属性)。普通属性的添加在<add>节中完成，开发人员必须为属性定义名称 name。

在默认情况下，用户配置功能仅存储经过验证的注册用户数据，不对匿名用户配置进行支持。因此，必须显示启用。在定义用户配置属性时，必须将其显示定义为可以由匿名用户单独使用。即在匿名用户可访问的属性中设置 allow Anonymous="true"。这样应用程序就会存储匿名用户的配置信息了。

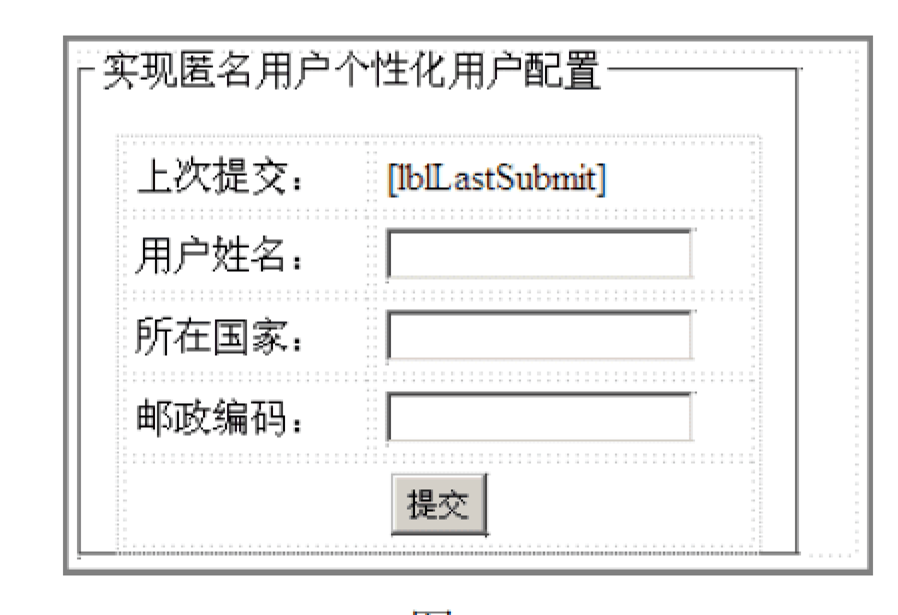


图 4-7

在 Default.aspx 页面的内容页中添加服务器控件、设置页面外观和布局等。界面如图 4-7 所示。

Default.aspx 页面的示例代码如下：

```
<form id="form1" runat="server">
```

```
<div>
<fieldset style="width: 300px">
    <legend class="mainTitle">实现匿名用户个性化用户配置</legend>
    <br />
    <table border="0" width="90%"   align="center" cellpadding="5">
        <tr>
            <td>上次提交：</td>
            <td>
            <asp:Label ID="lblLastSubmit" CssClass="commonText" runat="server"></asp:Label>
            </td>
        </tr>
        <tr>
            <td>用户姓名：</td>
            <td>
                <asp:TextBox ID="txtName" runat="server"></asp:TextBox>
            </td>
        </tr>
        <tr>
            <td>所在国家：</td>
            <td>
                <asp:TextBox ID="txtCity" runat="server"></asp:TextBox>
            </td>
        </tr>
        <tr>
            <td>邮政编码：</td>
            <td>
```



```
                <asp:TextBox ID="txtPostalCode" runat="server"></asp:TextBox>
            </td>
        </tr>
        <tr>
            <td colspan="2" align="center">
                <asp:Button ID="btnSubmit" runat="server" Text="提交"
                     OnClick="btnSubmit_Click" />
            </td>
        </tr>
    </table>
</fieldset>
</div>
</form>
```

(2) Default.aspx 文件实现了应用程序的外观，真正的核心代码存储在 Default.aspx.cs 文件中。该文件通过 Profile 属性实现了对个性化用户配置信息的访问和存储。完整的代码如下所示：

```
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
        {
            //显示用户配置信息
            DisplayProfileInfo();
        }
```

```
}
protected void btnSubmit_Click(object sender, EventArgs e)
{
    //保存用户配置信息到 Profile 属性中
    Profile.Name = txtName.Text;
    Profile.Address.City = txtCity.Text;
    Profile.Address.PostalCode = txtPostalCode.Text;
    Profile.LastSubmit = DateTime.Now;
    //显示用户配置信息
    DisplayProfileInfo();
}
private void DisplayProfileInfo()
{
    //从 Profile 属性中获取数据并赋值给服务器控件
    txtName.Text = Profile.Name;
    txtCity.Text = Profile.Address.City;
    txtPostalCode.Text = Profile.Address.PostalCode;
    DateTime time = Profile.LastSubmit;
    //如果未获取值则显示空，否则显示获取的值
    if (time.Year == 1)
    {
        labLastSubmit.Text = "空";
    }
    else
    {
        lblLastSubmit.Text = time.ToString();
```



```
        }
    }
}
```

如上所示，代码段主要包括两个部分，一个是页面加载的 Page_Load 事件处理程序，另一个是单击 Button 按钮所引发的 btnSubmint_Click 事件处理程序。Page_Load 事件处理程序中，调用了 DisplayProfileInfo 方法以显示从数据库获取的用户配置信息。btnSubmit_Click 的任务是将用户提交数据存储到数据库中。完成这两项任务使用了 Profile 属性。需要注意的是，从 Profile 属性获取用户配置属性的数据类型必须和 Web.Config 文件中<Profile>设置的属性的数据类型一致，默认为 String 类型。

(3) 编译保存，然后运行 Default.aspx 页面，如图 4-8 所示。

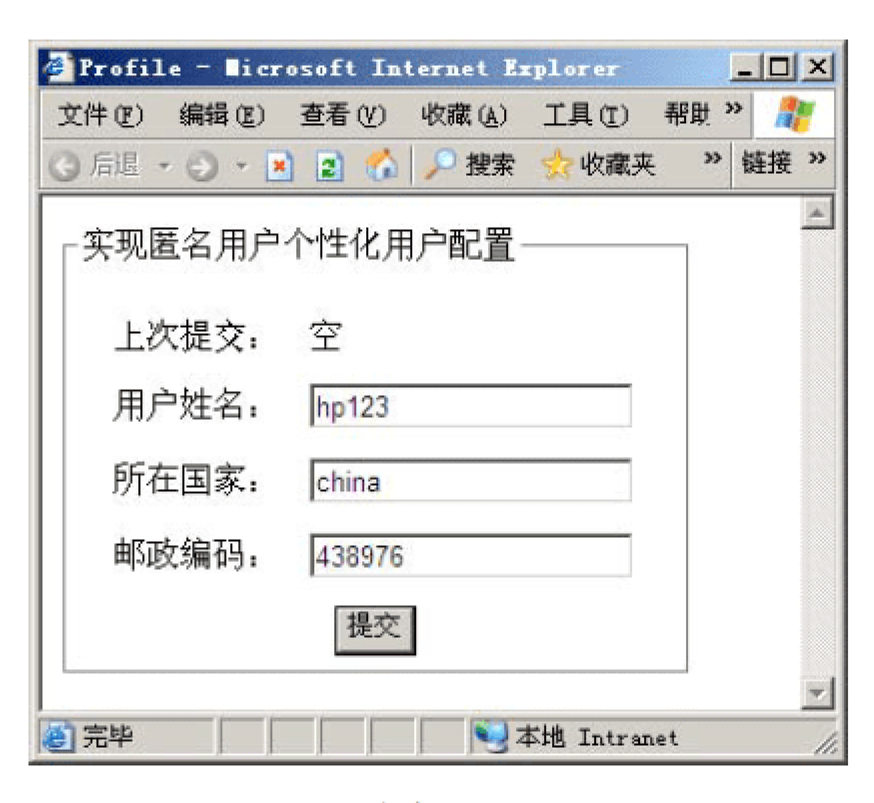


图 4-8

提交后如图 4-9 所示。

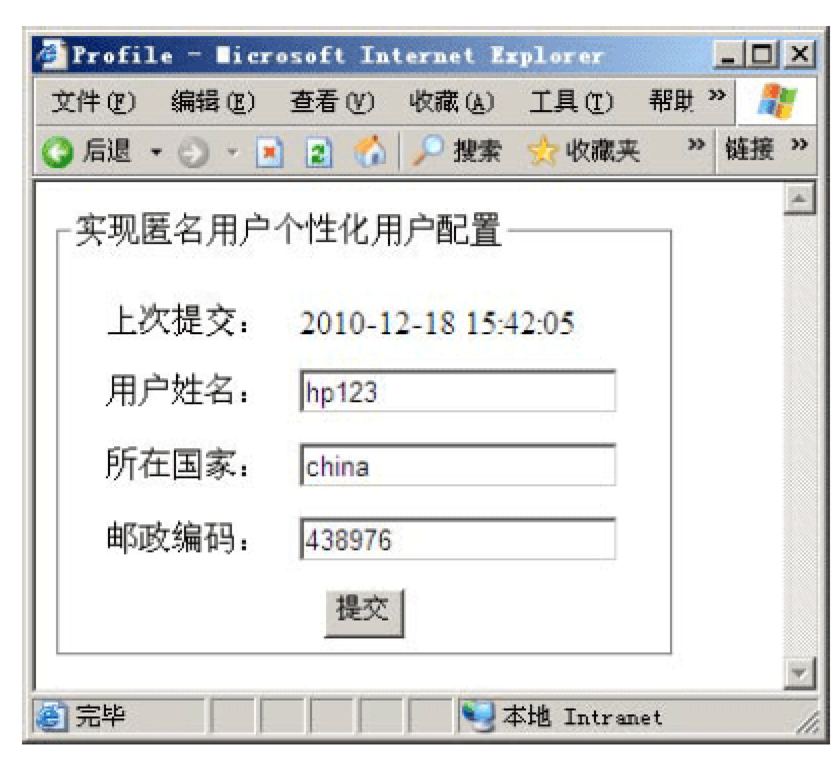


图 4-9

练习 2：显示所有注册用户个性化配置信息。

【问题描述】

某系统的用户信息包括用户名、密码、电子邮件、注册时间、性别和所在城市，考虑使用 ASP.NET 的便利性来开发系统，要求能够注册新用户，并且管理员可以查看用户的信息。

【问题分析】

注册新用户时需要提供性别和城市信息，将用户名、密码、电子邮件、注册时间信息用 Membership 来存储这些信息，性别和城市使用 Profile 来存储。管理员查看用户信息时分别从 Membership 和 Profile 取出数据进行展示。

【参考步骤】

(1) 创建一个网站，添加一个 Web.Config 文件，添加 Membership 和 Profile 配置节。

```
<connectionStrings>
  <clear/>
  <add name="constr"
       connectionString="server=.;database=aspnetdb;uid=sa;pwd=sa;"
       providerName="System.Data.SqlClient"/>
</connectionStrings>
<system.web>
<membership defaultProvider="SqlProvider">
  <providers>
    <clear />
    <add
      name="SqlProvider"
      type="System.Web.Security.SqlMembershipProvider"
      connectionStringName="constr"
```



```
        applicationName="cba"
        enablePasswordRetrieval="false"
        enablePasswordReset="true"
        requiresQuestionAndAnswer="false"
        requiresUniqueEmail="true"
        passwordFormat="Hashed" />
  </providers>
</membership>

<profile enabled="true">
  <providers>
    <clear/>
    <add name="AspNetSqlProfileProvider"
          connectionStringName="constr"
          applicationName="cba"
          type="System.Web.Profile.SqlProfileProvider"/>
  </providers>
  <properties>
    <add name="Sex"/>
    <add name="City"/>
  </properties>
</profile>
</system.web>
```

(2) 添加页面 AddUser.aspx，设计如图 4-10 所示。

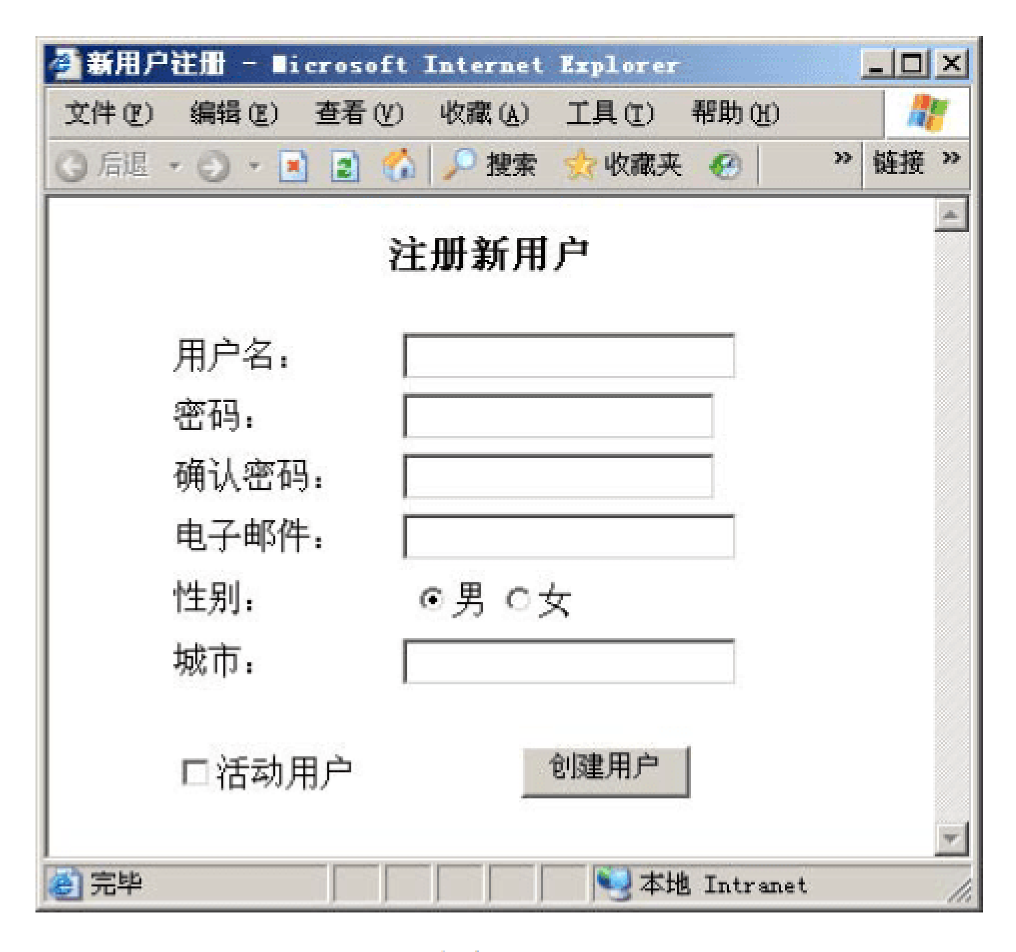


图 4-10

(3) 为“创建用户”按钮添加 Click 事件代码如下：

```
protected void btnAddUser_Click(object sender, EventArgs e)
{
    string name = this.txtName.Text;
    string pwd = this.txtPwd.Text;
    string email = this.txtEmail.Text;
    string sex = this.rblSex.SelectedValue;
    string city = this.txtCity.Text;
    bool active = this.chkActive.Checked;

    MembershipUser user = Membership.CreateUser(name, pwd, email);
    if (user != null)
    {
        user.IsApproved = active;
        //创建个性化配置信息
        ProfileCommon pc = (ProfileCommon)ProfileCommon.Create(name, true);
        pc.City = city;
```



```
            pc.Sex = sex;
            pc.Save();
        }
}
```

(4) 添加页面 UserList.aspx，使用 ObjectDataSource 和 GridView 显示用户信息，设计代码如下：

```
<asp:GridView ID="gv" runat="server"
    AutoGenerateColumns="False" DataKeyNames="UserName" DataSourceID="ds"
    onrowdatabound="gv_RowDataBound"
    onrowdeleting="gv_RowDeleting" Width="505px">
    <Columns>
        <asp:BoundField DataField="UserName" HeaderText="用户名"
            ReadOnly="True"/>
        <asp:BoundField DataField="Email" HeaderText="E-mail"   />
        <asp:BoundField HeaderText="性别" />
        <asp:BoundField HeaderText="所在城市" />
        <asp:CommandField ShowDeleteButton="True" ButtonType="Button" />
    </Columns>
</asp:GridView>
<asp:ObjectDataSource ID="ds" runat="server"
    SelectMethod="GetAllUsers" TypeName="System.Web.Security.Membership">
</asp:ObjectDataSource>
```

(5) 为 gv 添加 RowDataBound 事件，在该事件中获取当前行用户个性化信息并显示在 gv 中，代码如下：

```
protected void gv_RowDataBound(object sender, GridViewRowEventArgs e)
{
    if (e.Row.RowType == DataControlRowType.DataRow)
    {
        string username = e.Row.Cells[0].Text;
        //获取 username 个性化配置信息
        ProfileCommon pc = Profile.GetProfile(username);
        e.Row.Cells[2].Text = pc.Sex;
        e.Row.Cells[3].Text = pc.City;
    }
}
```

(6) 为 gv 添加 RowDeleting 事件，在该事件中删除用户及其个性化信息，代码如下：

```
protected void gv_RowDeleting(object sender, GridViewDeleteEventArgs e)
{
    string username = gv.Rows[e.RowIndex].Cells[0].Text;
    //删除用户
    Membership.DeleteUser(username, true);
    //删除个性化信息
System.Web.Profile.ProfileManager.DeleteProfile(username);

    e.Cancel = true;
    gv.DataBind();
}
```

(7) 运行网站，结果如图 4-11 所示。





图 4-11

◆ 第二阶段 ◆

练习：在第一阶段练习 2 的基础上实现更新注册用户信息功能。

【问题描述】

更新用户信息包括 Membership 信息和 Profile 信息。

【推荐步骤】

(1) 为 UserList.aspx 页面添加超链接列，超链接到 UpdateUser.aspx 页面。

(2) 添加 UpdateUser.aspx 页面，在 Load 事件中显示个性化信息。更新时保存个性化信息。

【课后作业】

在第二阶段练习的基础上，向注册用户个性化配置中添加一个名为 Contact 的 group 用来存储用户的多个联系方式。

第5章

数 据 缓 存



课程目标

- 缓存概述
- 页面输出缓存
- 页面部分缓存
- 应用程序缓存
- 缓存依赖
- 应用程序缓存移出回调





简 介

缓存是一种在计算机中广泛用来提高性能的技术，它将访问频率高或构造成本高的数据保留在内存中。在Web应用程序的上下文中，缓存用于在HTTP请求时保留页或者数据，并在无须新创建的情况下多次使用它们。请记住，一旦应用程序停止或者重新启动，其缓存将被清除。

生成高性能、可缩放的Web应用程序最重要的因素之一是能够在首次请求项时将这些项存储在内存中，不管它们是数据对象，还是页的某些部分。可以将这些项存储在Web服务器上或请求流中的其他软件上，例如代理服务器或浏览器。使用户能够避免重新创建满足先前请求的信息，尤其是那些需要大量处理器时间或资源的信息。这就是通常所说的缓存，它允许用户使用多种技术跨HTTP请求存储页输出或应用程序数据并对其进行重复使用。这样，服务器不需要重新创建信息，从而节省了时间和资源。

ASP.NET提供了两种可以用来创建高性能Web应用程序的缓存类型。第一种叫作输出缓存，它允许将动态页或用户控件响应存储在输出流(从发起服务器到请求浏览器)中任何具备HTTP 1.1缓存功能的设备上。当后面的请求发生时，不执行页或用户控件代码，缓存的输出用于满足该请求。第二种类型的缓存是传统的应用程序数据缓存，可以使用它以编程方式将任意对象(例如数据集)存储到服务器内存，这样用户的应用程序可以节省重新创建这些对象所需的时间和资源。

5.1 页面输出缓存

页面输出缓存作为最简单的缓存形式，将已经生成的动/静态页面全部内容保存在服务器内存中。当再有请求时，系统将缓存中的相关数据直接输出，直到缓存数据过期。这个过程中，缓存不需要再经过页面处理生命周期，这样可以缩短请求响应时间，提高应用程序性能。所以不难看出，页面输出缓存适用于不需要频繁更新数据，而占用大量时间和资

源才能编译生成的页面；对于数据经常更新的页面，则不适用。

设置页面输出缓存可以使用两种方式：

- @OutputCache 指令
- 页面输出缓存 API——HttpCachePolicy 类

5.1.1 @OutputCache 指令

ASP.NET 中的输出缓存可以使用服务器的内存将处理和显示页面的输出缓存起来。如果是在应用程序中使用页面输出缓存机制，此页面第一次被请求时，页面输出的数据就会被缓存；此时如果该页面再被请求，服务器只是把内存中的页面副本(缓存的页面输出数据)发送至客户端，而不再需要系统重复去分析、编译和处理该页面请求。这就在很大程度上提高了应用程序的响应时间，同时也减少了对服务器资源的使用。

启用页面缓存机制的方法非常简单，只需要在页面的顶部添加一个@OutputCache 指令，就可以非常容易地将缓存页面输出。例如，下面的语句将页面输出以最长 60 秒时间进行缓存：

```
<%@ OutputCache Duration="60" VaryByParam="none" %>
```

该指令接受一组属性，其中 Duration 和 VaryByParam 两个属性是必需的。Duration 属性表示系统缓存页面输出的缓存时间(单位：秒)。VaryByParam 属性允许根据 GET 查询字符串或者 POST 形参，改变所要缓存的输出。表 5-1 列出了@OutputCache 指令的常用属性。

表 5-1 @OutputCache 指令的常用属性

属　性	适用对象	描　述
Duration	页面和用户控件	页面或用户控件的缓存时间(单位：秒)
Location	页面	为存储页面的输出指定一个有效的位置
Shared	用户控件	一个布尔值，确定用户控件输出是否可以由多个页共享。默认值为 false



（续表）

属　　性	适用对象	描　　述
SqlDependency	页面和用户控件	表示一个给定 SQL Server 数据库中某个指定的依赖对象。只要修改该表的内容，就将从缓存中删除页面输出
VaryByControl	用户控件	一个由分号分隔的字符串列表，用于更改用户控件的输出缓存。这些字符串代表用户控件中声明的 ASP.NET 服务器控件的 ID 属性值
VaryByCustom	页面和用户控件	表示自定义输出缓存要求任意文本。如果赋予该属性值为 browser，缓存将随浏览器名称和主要版本信息的不同而异。如果输入自定义字符串，则必须在应用程序 Global.asax 文件重写 GetVaryByCustomString
VaryByParam	页面和用户控件	分号分隔的字符串列表，用于使输出缓存发生变化。默认情况下，这些字符串与随 GET 方法属性发送的查询字符串值对应，或与使用 POST 方法发送的参数对应。将该属性设置为多个参数时，对于每个指定参数组合，输出缓存都包含一个不同版本的请求文档。可能的值包括 none、星号(*)以及任何有效的查询字符串或 POST 参数名称

5.1.2　HttpCachePolicy 类

使用@OutputCache 指令实现对于输出缓存的各项设置，这种方法简单易行。同时还可以使用输出缓存 API 来编程设置页面输出缓存。这种使用编程来设置页面输出缓存的方法的核心是调用 System.Web.HttpCachePolicy。该类主要包含用于设置缓存特定的 HTTP 标头的方法和用于控制 ASP.NET 页面输出缓存的方法。

下面的代码说明了页面缓存 API 的 HttpCachePolicy 类的使用方法：

```
Response.Cache.SetExpires(DateTime.Now.AddSeconds(30));
Response.Cache.SetExpires(DateTime.Parse("19:00:00"));
```

Response 类的 Cache 属性用于获取页面缓存策略，该属性的数据类型是 HttpCachePolicy。可以通过调用 Response.Cache 来获取 HttpCachePolicy 实例，进而实现对

于当前页面输出缓存的设置。在上面所示的代码中，第一行代码表示输出缓存时间是 30s，并且页面不随任何 GET 或 POST 参数改变，相当于在页面头加入指令<%@ OutputCache Duration="30" VaryByParam="none" %>。第二行代码设置缓存过期的绝对时间是当日下午 7 点整。

5.2 页面部分缓存

页面部分缓存是指输出缓存页面的某些部分，而不是缓存整个页面内容。实现页面部分缓存有两种机制：一种是将页面中需要缓存的部分置于用户控件(.ascx 文件)中，并且为用户控件设置缓存功能，这就是“控件缓存”。设置控件缓存的实质是对用户控件进行缓存配置。另一种是“缓存后替换”，该方法与控件缓存正好相反，将页面中的某一部分设置为不缓存，因此，尽管缓存了整个页面，但是当再次请求该页时，将重新处理那些没有设置为缓存的内容。

5.2.1 控件缓存

控件缓存是通过用户控件来实现局部页面缓存机制的。控件缓存类似于页面输出缓存，它们使用相同的指令@OutputCache。控件缓存与页面输出缓存的@OutputCache指令既有相同之处，也有不同的方面。两者的共同点在于它们的设置方法基本相同，都是在文件顶部设置包含属性的@OutputCache 指令字符串。不同点在于：一是控件缓存的@OutputCahce 指令设置在用户控件文件(.ascx)中，页面输出缓存的@OutputCache指令设置在普通页面(.aspx)中。二是控件缓存的@OutputCache 指令只能设置 Duration、VaryByParam、VaryByControl、VaryByCustom、Shared 和 SqlDependency 等 6 个属性，而页面输出缓存的@OutputCache 指令可设置所有的属性。

利用@OutputCache 指令设置控件缓存的常见写法如下所示：

```
<%@ OutputCache Duration="60" VaryByParam="Id;CategoryId" %>
```



以上代码表明设置用户控件缓存有效时间是 60 秒，并且允许使用 Id 和 CategoryId 参数来改变缓存。通过 VaryByParam 属性设置，在服务器缓存中可能存储多个用户控件的不同实例。例如，对于一个包含该用户控件的页面，有如下两个请求：

```
http://localhost:8632/Pubs/AuthorList.aspx?id=1&CategoryId=1
http://localhost:8632/Pubs/AuthorList.aspx?id=1&CategoryId=2
```

当请求的页面如上时，由于控件中@OutputCache 指令的属性 VaryByParam 值被设置为“Id;CategoryId”，那么在服务器缓存中就会存储两个版本的用户控件缓存实例。

控件缓存设置除了支持以上所述 VaryByParam 属性外，还支持 VaryByControl 属性。VaryByParam 属性基于使用 Post 或者 Get 方式发送的名称/值对来改变缓存，而 VaryByControl 属性通过用户控件文件中包含的服务器控件来改变缓存。代码如下所示：

```
<%@ OutputCache Duration="60" VaryByParam="none" VaryByControl="ddlGood" %>
```

以上代码设置缓存有效期是 60 秒，并且页面不随任何 Get 或 Post 参数改变。如果用户控件中包含 Id 属性为 ddlGood 的服务器控件，那么缓存将根据该控件的变化来存储用户控件数据。

5.2.2　缓存后替换

ASP.NET 页面中既包含静态内容，又包含基于数据库数据的动态内容。静态内容通常不会发生变化。因此，对静态内容实现数据缓存是非常有必要的。然而，那些基于数据的动态内容则不同。数据库中的数据可能每时每刻都发生变化，因此，如果对动态内容实现缓存，可能造成数据不能及时更新的问题。

如何才能够实现缓存页面的大部分内容，而不缓存页面中的某些片段？显然，用前面讲到的缓存机制不太容易实现，ASP.NET 提供了一种缓存后替换的功能。

在实际的应用中，有时为了提高应用程序的性能，通常会去缓存整个 ASP.NET 页面，同时，需要动态更新该页上与时间或者用户高度相关的信息。这时可以使用 ASP.NET 提

供的 Substitution 控件。Substitution 控件能够指定页面输出缓存中需要以动态内容替换该控件的部分，即允许对整个页面进行输出缓存，然后，使用 Substitution 控件指定页中免于缓存的部分。需要缓存的区域只执行一次，然后从缓存读取，直至该缓存项到期或被清除。动态区域，也就是 Substitution 控件指定的部分，在每次请求页面时都执行。

在使用 Substitution 时，首先将整个页面缓存起来，然后将页面中需要动态改变内容的地方用 Substitution 控件代替即可。Substitution 控件需要设置一个重要属性 MethodName，该属性用于获取或者设置当 Substitution 控件执行时为回调而调用的方法名称。该方法必须要符合 3 点：

- 该方法必须被定义为静态方法。
- 该方法必须接受 HttpContext 类型的参数。
- 该方法必须返回 String 类型的值。

在运行情况下，Substitution 控件将自动调用 MethodName 属性所定义的方法。该方法返回的字符串，即为要在页面中的 Substitution 控件的位置上显示的内容。当第一次请求页面时，该页将运行并缓存其输出。对于后续的请求，在页面缓存过期前，只有 Substitution 控件及其有关方法在每次请求时执行，并且自动更新该控件所表示的动态内容。

下面来实现一个使用 Substitution 控件完成缓存后替换功能的实例：

(1) 打开 Visual Studio 2008，新建网站 Example5_1。

(2) 打开页面 Default.aspx，在页面代码中添加@OutputCache 指令，代码如下：

```
<%@ OutputCache Duration="30" VaryByParam="none" %>
```

(3) 在页面中加入服务器控件，完成布局。代码如下：

```
<form id="form1" runat="server">
    <div>
        <fieldset>
            <legend>使用 Substitution 控件实现缓存后替换</legend><br />
            <div>以下时间显示，使用了 Substitution 控件</div>
```



```
            <asp:Substitution ID="SubTime" MethodName="GetTime" runat="server" />
            <hr style="color: #FF0000" />
            <div>以下时间显示为页面缓存，缓存时间为 30 秒</div>
            <asp:Label ID="labTime" runat="server"></asp:Label>
            <br />
        </fieldset>
</div>
</form>
```

(4) 在后台代码中，添加 GetTime 方法，完成时间的显示。完整的代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    this.labTime.Text = DateTime.Now.ToLongTimeString();
}

public static string GetTime(HttpContext context)
{
    return DateTime.Now.ToLongTimeString();
}
```

(5) 保存代码，运行调试。运行结果如图 5-1 所示。

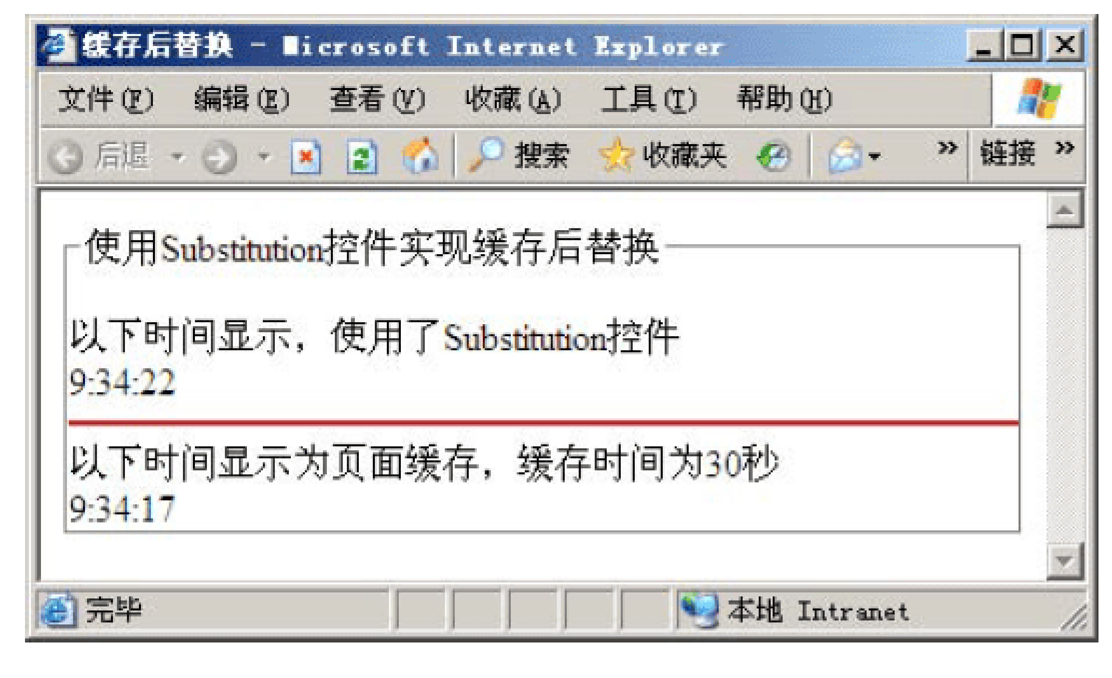


图 5-1

在示例中页面主要包括 Substitution、Label 两个控件。在 Page_Load 事件处理程序中设置了 Label 控件显示时间值。还实现了一个静态方法 GetTime，该方法带有一个 HttpContext 类型的参数，返回值为 String 类型，其返回内容为当前时间。在代码顶部使用 @OutputCache 指令设置页面输出缓存过期时间为 30 秒，将整个页面都缓存起来。所以当单击浏览器的刷新按钮时，会发现 Label 控件所显示的时间值没有发生变化，每当间隔 30 秒后才会改变为当前时间。Substitution 控件的 MethodName 属性值为 GetTime，表明该控件显示的内容来自于 GetTime 方法的返回值。虽然将整个页面设置了缓存输出，但是每当页面刷新时，Substitution 控件仍然被重新执行，并将 MethodName 属性值指定的方法返回值显示在页面上。因此，显示的总是当前时间。

5.3 应用程序数据缓存

页面输出缓存和页面部分缓存可以将页面整体或者其中的部分存储到内存中，下面介绍应用程序数据缓存方面的内容。

应用程序数据缓存的主要功能是在内存中存储各种与应用程序相关的对象。通常这些对象都需要耗费大量的服务器资源才能创建。因此，对这些对象实施缓存，无论是对服务器还是对用户都有着明显的好处。

应用程序数据缓存是易失的，当服务器内存不足时系统会自动让缓存过期，所以使用缓存存储较多的数据时必须保证 Web 服务器的内存足够大。

5.3.1 Cache 类

应用程序数据缓存由 Cache 类实现，该类从属于 System.Web.Caching 命名空间，其实例对象为每个应用程序所专用。通过 Cache 类的应用，可实现添加、检索和移除应用程序数据缓存，以及移除缓存项时通知应用程序等功能。当应用程序启动时，系统自动为每个应用程序创建该类的一个实例，该类对象在应用程序生命周期中都有效。Cache 类使用的



语法与会话状态和应用程序状态使用的语法类似，对象按“键/值”对保存在缓存的字典对象中，保存的对象是值，而键则是一个描述性字符串。下面是一个简单的对货品编号值，实现缓存的代码：

```
Cache["CategoryId"] = Request.QueryString["CategoryId"];
```

上面的代码通过指定键和值，向应用程序数据缓存添加了一个新的缓存数据项。使用这种方法，不仅可以缓存简单数据对象，还可以缓存如 DataView、DataSet 等复杂数据对象。

表 5-2 列出了 Cache 类的主要方法。

表 5-2　Cache 类的主要方法

方　法	描　述
Add	将指定项添加到 Cache 对象，该对象具有依赖项、过期和优先级策略以及一个委托(可用于在从 Cache 移除插入项时通知应用程序)
Insert	向 Cache 对象插入项。使用此方法的某一版本改写具有相同 key 参数的现有 Cache 项
Remove	从应用程序的 Cache 对象移除指定项

5.3.2　Add 方法

Add 方法是 Cache 类的重要方法之一，该方法在将数据项添加到缓存的同时，还允许为应用程序数据缓存设置有效期、优先级、依赖项等特性。下面的代码列出了 Add 方法的声明：

```
public Object Add(string key,
    Object value,
    CacheDependency dependencies,
    DateTime absoluteExpiration,
    TimeSpan slidingExpiration,
```

```
    CacheItemPriority priority,
CacheItemRemovedCallback onRemoveCallback)
```

下面对上述代码进行说明。

- Key：用于引用该项的缓存键。
- Value：要添加到缓存的项。
- Dependencies：该项的文件依赖项或缓存键依赖项。当任何依赖项更改时，该对象即无效，并从缓存中移除。如果没有依赖项，则此参数包含 null 引用。
- absoluteExpiration：所添加对象将过期并被从缓存中移除。如果使用可调过期，则 absoluteExpiration 参数必须为 NoAbsoluteExpiration。
- slidingExpiration：最后一次访问所添加对象时与该对象过期时之间的时间间隔。如果该值等效于 20 分钟，则对象在最后一次被访问 20 分钟之后将过期并从缓存中移除。如果使用绝对过期，则 slidingExpiration 参数必须为 NoSlidingExpiration。
- priority：对象的相对成本，由 CacheItemPriority 枚举表示。缓存在退出对象时使用该值；具有较低成本的对象在具有较高成本的对象之前被从缓存中移除。
- onRemoveCallback：在从缓存中移除对象时所调用的委托(如果提供)。当从缓存中删除应用程序的对象时，可使用它来通知应用程序。

在了解了 Add 方法的参数后，就可以使用该方法了。下面的示例创建了一个 AddItemToCache 方法。该方法检查与缓存中的 Key1 键关联的值。如果该值为 null 引用，则 Add 方法在缓存中放置一项，其键为 Key1，值为 Value1，绝对过期时间为 60 秒，并具有高缓存优先级。

```
public void AddItemToCache(Object sender, EventArgs e)
{
    if (Cache["Key1"] == null)
    {
        Cache.Add("Key1", "Value 1", null, DateTime.Now.AddSeconds(60),
```



```
                Cache.NoSlidingExpiration, CacheItemPriority.High, null);
    }
}
```

在使用 Add 方法的时候，需要注意的是，如果缓存中已保存了具有相同键名的项，则对此方法的调用将失败。若要使用相同的 key 参数改写现有的 Cache 项，那么必须使用 Insert 方法。

5.3.3 Insert 方法

Insert 方法与 Add 方法具有相似之处。例如，两者所实现的功能基本相同，都可用于添加页面数据对象。但是 Insert 方法支持 4 种重载方式(参数说明与 Add 方法参数基本相同)：

```
Insert(String, Object)
Insert(String, Object, CacheDependency)
Insert(String, Object, CacheDependency, DateTime, TimeSpan)
Insert(String, Object, CacheDependency, DateTime, TimeSpan, CacheItemPriority,
CacheItemRemovedCallback)
```

Add 方法在使用上没有 Insert 方法那样灵活，使用 Add 方法时必须提供 7 个参数，而使用 Insert 方法则可根据需要随意选取重载方式。另外，如果调用 Insert 或者 Add 方法，将已存储在缓存中的某项添加到重复缓存中时，那么它们的策略是不同的。Insert 方法替换该选项，而 Add 方法报告失败。

下面列举 Insert 方法的常用示例：

```
//代码示例将有一分钟绝对过期时间的项添加到缓存中
Cache.Insert("CacheItem6", "Cached Item 6",
    null, DateTime.Now.AddMinutes(1d),
```

```
System.Web.Caching.Cache.NoSlidingExpiration);
```

5.3.4 检索应用程序缓存对象

从缓存中检索应用程序数据缓存对象，通常可以使用两种方法。

- 指定键名。
- 使用 Cache 类的 Get 方法。

由于缓存中所存储的信息为易失信息，即该信息可能由 ASP.NET 移除。因此，在平日的开发中，通常是首先确定该项是否在缓存中。如果不在，则应将它重新添加到缓存中，然后检索该项。下面具体介绍检索应用程序缓存对象的 2 种方法。

(1) 通过指定键名，实现应用程序数据缓存对象的检索

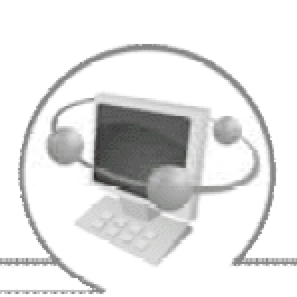
使用这种方法时，只需要指定表示数据对象的键名，就可以获得其对应的数据对象。代码段如下所示：

```
if (Cache["categoryId"] != null)
{
string categoryId = Cache["categoryId"].ToString();
...
}
```

在使用该种方式来检索应用程序数据缓存对象时，要注意两个问题：一是获取的数据对象都是 Object 类型，所以必须进行相关的强制类型转换；二是在使用转换后的数据对象时，首先要判断是否为空，然后再执行相应的其他操作。

(2) 通过使用 Get 方法，实现应用程序数据缓存对象的检索

Get 方法是 Cache 类的方法之一，其功能是根据键名获取被缓存的数据对象。该方法通过设置表示键名的参数 Key 来获取被缓存的数据对象，其返回值是 Object 类型，所以在使用的过程中也需要注意类型转换。代码段如下所示：



```
if (Cache["categoryId"] != null)
{
string categoryId = Cache.Get("categoryId").ToString();
...
}
```

5.4 缓存依赖

缓存依赖是实现缓存功能中非常重要的部分。通过缓存依赖，可以在被依赖对象(如文件、目录、数据库表等)与缓存对象之间建立一个有效关联。当被依赖对象发生变化时，缓存对象将变得不可用，并被自动从缓存中移除。假设有一个包含奥运奖牌榜数据的 XML 文件，该文件随着奥运赛事的进行，被不定期地修改。应用程序的功能是利用从该文件中读取的数据，构造一个奥运奖牌榜表格，以显示当前奖牌的分布情况。程序在首次读取 XML 文件时，将数据置入缓存中，并记录下缓存对象与文件之间的依赖关系。当 XML 文件被修改时，由于依赖关系的作用，因此，开始生成的缓存对象将被从缓存中移除。然后，再重新读取文件并创建新的缓存对象。这样就完成了缓存更新。

通过缓存依赖的设置既提高了应用程序的响应，减轻了服务器的负担，同时也让用户浏览时看到的数据总是实时的、最新的。如图 5-2 所示是实现缓存依赖功能的类层次结构。

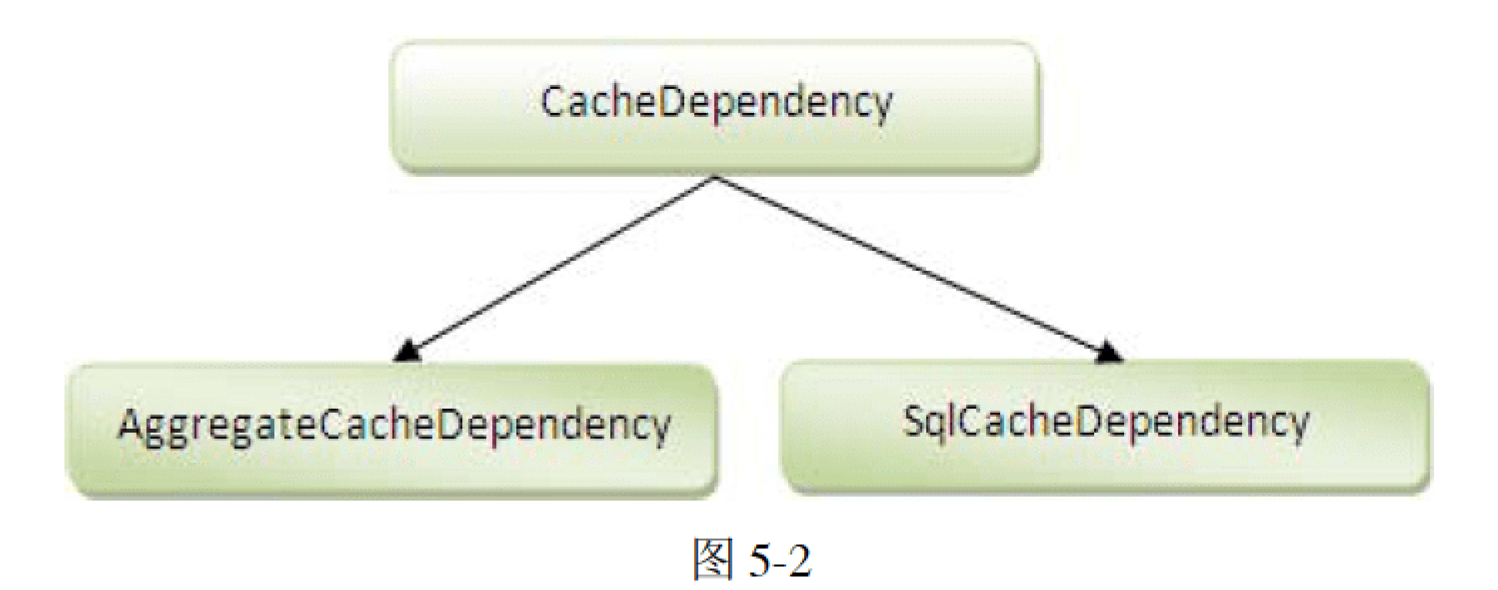


图 5-2

5.4.1 CacheDependency 类

由图 5-2 可知，CacheDependency 是实现缓存依赖功能的核心类之一，其主要功能是在 ASP.NET 应用程序数据缓存对象与文件、缓存键、文件或缓存键的数组或另一个 CacheDependency 对象之间，建立一种依赖关系。通过 CacheDependency 类创建的缓存依赖称为自定义缓存依赖。

当添加一个应用程序数据缓存对象，并通过 CacheDependency 类设置缓存依赖关系后，系统将自动创建一个 CacheDependency 实例，来跟踪对所指定文件、目录或其他缓存依赖对象的更改。当这些被依赖对象发生更改时，缓存对象则自动被移除。例如，假设创建了一个基于 XML 文件的 DataSet 对象，并且该对象被作为应用程序数据缓存对象添加时，使用 CacheDependency 类，在 DataSet 对象与 XML 文件之间建立了依赖关系，那么，当 XML 文件被修改时，DataSet 的缓存对象将被从缓存中自动移除。通常情况下，在添加自定义缓存依赖过程中，首先要创建 CacheDependency 类实例，然后再使用 Cache 类的 Insert 或 Add 方法实现缓存对象的添加。代码段如下所示：

```
//使用 CacheDependency 类添加具有缓存依赖的缓存对象
string fileName = Server.MapPath("sale.xml");

//创建缓存依赖
System.Web.Caching.CacheDependency dep =
        new System.Web.Caching.CacheDependency(fileName, DateTime.Now);
//添加应用程序数据缓存
Cache.Insert("key", "value", dep);
```

首先，使用 CacheDependency 构造函数实例化 CacheDependency 对象，然后，利用 Insert 方法将这个具有缓存依赖关系的数据对象添加到缓存中。在创建缓存依赖实例 dep 的过程中，设置缓存对象所依赖的文件路径是 fileName。当更改该文件时，缓存对象将过期，并从缓存中移除。



下面通过一个示例来说明使用 CacheDependency 类实现自定义缓存依赖的方法。

(1) 打开 Visual Studio 2008，新建一网站 Example5_2。

(2) 添加 XML 文件，重命名为 sale.xml。

(3) 打开 sale.xml 文件，添加测试数据。代码如下：

```
<?xml version="1.0" encoding="gb2312"?>
<Goods>
  <Item Breed="格力" Type="KFR-32GW/R(23540)-N5" Number="1500" />
  <Item Breed="格力" Type="KFR-23GW/R(23540)-N5" Number="800" />
  <Item Breed="美的" Type="KFR-23GW/DY-GA(E5)" Number="1000" />
</Goods>
```

(4) 打开默认页面 Default.aspx，在页面中添加服务器控件。页面布局如图 5-3 所示。

自定义缓存依赖应用

Column0	Column1	Column2
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc

品牌：　型号：　数量：

添加数据

图 5-3

代码如下所示：

```
<form id="form1" runat="server">
    <div>
        <fieldset style="width: 398px">
            <legend>自定义缓存依赖应用</legend><br />
            <div>
```

```
            <center>
                <asp:GridView ID="gvSale" runat="server" BackColor="White"
                    BorderColor="White" BorderStyle="Ridge"
                    BorderWidth="2px" CellPadding="3"
                    GridLines="None" Width="100%" CellSpacing="1">
                    <FooterStyle BackColor="#C6C3C6" ForeColor="Black" />
                    <RowStyle BackColor="#DEDFDE" ForeColor="Black" />
                    <PagerStyle BackColor="#C6C3C6" ForeColor="Black"
                        HorizontalAlign="Right" />
                    <SelectedRowStyle BackColor="#9471DE" Font-Bold="True"
                        ForeColor="White" />
                    <HeaderStyle BackColor="#4A3C8C" Font-Bold="True"
                        ForeColor="#E7E7FF" />
                </asp:GridView>
            </center>
        </div>
        <div>
            <center>
                <table class="style1">
                    <tr>
                        <td>
品牌：<asp:TextBox ID="txtBreed" runat="server" Width="70px"></asp:TextBox>
                        </td>
                        <td>
型号：<asp:TextBox ID="txtType" runat="server" Width="70px"></asp:TextBox>
                        </td>
```



```
                                        <td>
数量：<asp:TextBox ID="txtNumber" runat="server" Width="70px"></asp:TextBox>
                                        </td>
                                    </tr>
                                    <tr>
                                        <td colspan="3">
<asp:Button ID="btnAdd" runat="server" Text="添加数据" onclick="btnAdd_Click" />
                                        </td>
                                    </tr>
                                </table>
                            </center>
                            <div>
                                <hr style="width: 394px" />
                            </div>
                        </div>
                    </fieldset>
</div>
</form>
```

页面 Default.aspx 的后台代码如下所示：

```
protected void Page_Load(object sender, EventArgs e)
    {
        if (!this.IsPostBack)
        {
            //加载 XML 文档中的数据
            LoadData();
```

```
        }
    }

public void LoadData()
{
    DataSet ds;
    //如果缓存为空，则添加缓存
    if (Cache["SaleData"] == null)
    {
        ds = new DataSet();
        string filePath = Server.MapPath("sale.xml");
        ds.ReadXml(filePath);
        //定义缓存依赖
        CacheDependency dep = new CacheDependency(filePath, DateTime.Now);
        Cache.Insert("SaleData", ds, dep);
    }
    else
    {
        ds = (DataSet)(Cache["SaleData"]);
    }
    this.gvSale.DataSource = ds;
    this.gvSale.DataBind();
}

protected void btnAdd_Click(object sender, EventArgs e)
{
```



```
    //读取 XML 文件数据
    XmlDocument saleXml = new XmlDocument();
    saleXml.Load(Server.MapPath("sale.xml"));
    //创建一个新的节点
    XmlElement breed = saleXml.CreateElement("Item");
    breed.SetAttribute("Breed", this.txtBreed.Text);
    breed.SetAttribute("Type", this.txtType.Text);
    breed.SetAttribute("Number", this.txtNumber.Text);
    //将新节点加入到 XML 中
    saleXml.DocumentElement.AppendChild(breed);
    //将添加的数据保存到文件中
    saleXml.Save(Server.MapPath("sale.xml"));


    Cache.Remove("SaleData");
    this.LoadData();
}
```

(5) 保存并运行，其结果如图 5-4 所示。



图 5-4

首次加载应用程序时，页面通过 GridView 控件以表格形式显示了一个 XML 文件的数据。当用户在 3 个文本框中任意添加字符串，并单击“添加数据”按钮后，页面将发生回传，并显示如图 5-4 所示的内容。用户所添加内容已显示在表格中。

页面加载的主要任务是显示数据，其通过调用 LoadData 方法来完成。在 LoadData 方法中，首先判断是否存在缓存对象。如果不存在则自行创建一个缓存对象。如果存在，则直接调用 GridView 控件显示缓存对象内容。此处的核心是创建缓存对象过程。首先，程序调用 ReadXml 方法，读取 XML 文件数据，然后，使用 CacheDependency 类设置缓存对象与 XML 文件之间的依赖关系，最后，利用 Cache 类的 Insert 方法将这种依赖关系添加到缓存对象中。

实现以上示例的关键是在数据缓存对象与 XML 文件之间建立缓存依赖关系。单击“添加数据”按钮的实质是将输入内容添加到 XML 文件中，由此达到修改 XML 文件的目的。由于缓存依赖关系的作用，缓存对象将被移除。

5.4.2 实现 SQL 数据缓存依赖

SQL 数据缓存依赖功能的核心是利用 SqlCacheDependency 类，在应用程序数据缓存对象与 SQL Server 数据库表，或者 SQL Server 查询结果之间，建立一种缓存依赖关系。由 SqlCacheDependency 对象监测数据表行变化情况。如果其中数据发生变化，将自动移除缓存对象。当下次请求该缓存对象时，如果该对象不在缓存中，可以再次向缓存中添加更新后的版本，并且可确保具有最新的数据。

实现 SQL 数据缓存依赖功能设置主要是通过应用 SqlCacheDependency 类，该类继承自 CacheDependency 类。可以通过该类实现 SQL 数据缓存依赖的初始化和设置等任务。具体而言，通过该类在应用程序数据缓存对象与 SQL Server 数据库表、数据行之间建立缓存依赖关系。

SqlCacheDependency 类包含的属性、方法绝大多数从父类继承，因此，要了解 SqlCacheDependency 类的构造函数。下面列举了该类构造函数代码：

```
public SqlCacheDependency(SqlCommand sqlcmd);
```



```
public SqlCacheDependency(string databaseEntryName,string tableName);
```

如上代码所示，SqlCacheDependency 类包含两个构造函数。这意味着可以使用两种方式初始化 SqlCacheDependency 类。

在构造 SQL 数据缓存依赖对象时，要注意：应用 SQL Server 时，必须使用构造函数一，在 sqlcmd 中将涉及相关的 SQL 查询语句(select)，这些语句必须满足以下要求：

- 必须定义完全限定的表名，包括表所有者的名称，如 dbo.Users。
- 必须在 select 语名中显示指定列名。不能使用星号(*)通配符来选择表中的所有列。例如，不能使用“select * from Users”，而必须使用“select name,address,email from Users”。
- 不能在查询语句中使用聚合函数。
- 必须为当前数据库启动 SQL Server Service Broker。如果当前数据库没有启用 Broker，可以执行“ALTER DATABASE 数据库名 SET ENABLE_BROKER”来启用。

构造函数二，是为使用 SQL Server 2005 以前版本而设置的，其中一个参数引用 web.config 文件中<sqlCacheDependency>的<database>子配置节中定义的数据库名称。

下面就通过一个示例来演示一下如何在页面输出缓存中启用 SQL 数据缓存依赖。具体操作如下。

(1) 打开 Visual Studio 2008，新建一个网站，命名为 Example5_3。

(2) 在 SQL Server 中实现数据缓存依赖，需要显式调用 SqlDependency 的 Start 方法。该方法用于启动接收依赖项更改通知的侦听器。在应用程序中，只需调用一次这个方法即可。通常，在 global.asax 文件的 Application_Start()方法中实现调用。打开 global.asax，开启 SQL Server 的数据缓存依赖，代码如下：

```
void Application_Start(object sender, EventArgs e)
{
    //在应用程序启动时运行的代码
```

```
    System.Data.SqlClient.SqlDependency.Start(
        System.Configuration.ConfigurationManager.
        ConnectionStrings["pub"].ConnectionString);
}
```

从上面的代码段中可以看出 Start 方法的一个参数是数据库连接字符串，将其保存在配置文件 web.config 中。

(3) 启用页面输出缓存，添加@OutputCache 指令：

```
<%@ OutputCache Duration="100" SqlDependency="CommandNotification" VaryByParam="none" %>
```

其中，SqlDependency 属性的值为 CommandNotification，表示当前页使用基于通知传递服务的 SQL 数据缓存依赖。

(4) 在默认页面 Default.aspx 中，放入一个 Lable 和一个 GridView，代码如下：

```
<form id="form1" runat="server">
    <div>
        <asp:Label ID="lblTime" runat="server" ForeColor="Red"
            Font-Size="XX-Large" Text=""></asp:Label>
        <asp:GridView ID="gvData" runat="server" BackColor="White"
            BorderColor="White" BorderStyle="Ridge" BorderWidth="2px"
            CellPadding="3" CellSpacing="1" GridLines="None">
             <FooterStyle BackColor="#C6C3C6" ForeColor="Black" />
             <RowStyle BackColor="#DEDFDE" ForeColor="Black" />
             <PagerStyle BackColor="#C6C3C6" ForeColor="Black"
                 HorizontalAlign="Right" />
             <SelectedRowStyle BackColor="#9471DE" Font-Bold="True"
                 ForeColor="White" />
```



```
                    <HeaderStyle BackColor="#4A3C8C" Font-Bold="True"
ForeColor="#E7E7FF" />
              </asp:GridView>
</div>
</form>
```

(5) 在 Default.aspx.cs 中加入后台代码，绑定显式数据。代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
       if (!this.IsPostBack)
       {
              this.lblTime.Text = "时间：" + DateTime.Now.ToLongTimeString();
              this.Bind();
       }
}
public void Bind()
{
SqlConnection conn = new SqlConnection(
                            ConfigurationManager.ConnectionStrings["pub"].ConnectionString);
SqlCommand cmd = new SqlCommand(
                            "SELECT [discounttype], [stor_id], [lowqty], [highqty], [discount]
                            FROM dbo.[discounts] ", conn);
       SqlDataAdapter da = new SqlDataAdapter(cmd);
       DataSet ds = new DataSet();
       da.Fill(ds);
```

```
        this.gvData.DataSource = ds;
        this.gvData.DataBind();
}
```

(6) 保存并运行，其结果如图 5-5 所示。

SQL数据库缓存依赖 - Microsoft Internet Explorer

时间：11:26:34

discounttype	stor_id	lowqty	highqty	discount
Customer Discount	6380		100	5.00
Initial Customer	7066	10	100	10.50
VIP Discount	6380	12	500	15.00
VIP Discount	7066	10	500	13.00
VIP Discount	7067	10	50	5.00

图 5-5

直接打开 pubs 数据库，修改 discounts 表，删除后三行，将第一行的 lowqty 设为 50，保存后刷新界面如图 5-6 所示。



discounttype	stor_id	lowqty	highqty	discount
Customer Discount	6380	50	100	5.00
Initial Customer	7066	10	100	10.50

图 5-6

页面首次运行时，显示了 discounts 数据表中的数据和页面刷新时间。由于利用@OutputCache 指令设置了页面输出缓存(缓存有效时间为 100 秒)，因此，100 秒内无论如何刷新页面，表格数据和页面刷新时间内容都不应该发生变化。该页面还同时使用了 SQL 数据缓存依赖功能，那么当修改 discounts 数据表中的数据，并再次刷新页面时，其结果如图 5-6 所示。这种变化就是由于 SQL 数据缓存依赖功能的作用。每当数据表发生变化，则自动移除原有页面输出缓存内容，实施缓存数据更新。



5.4.3　聚合缓存依赖 AggregateCacheDependency 类

由图 5-2 可知，AggregateCacheDependency 继承自 CacheDependency，属于封装类，不可被继承。该类的主要功能是实现聚合缓存依赖功能。具体来说，它可以在单个应用程序缓存对象与多个依赖项之间建立缓存依赖关系。多个缓存依赖项实际是 CacheDependency 对象组，该组中的对象可以是 CacheDependency 对象、SqlCacheDependency 对象、继承自 CacheDependency 类的自定义缓存依赖对象，或者是以上三者的任意组合等。AggregateCacheDependency 类负责跟踪 CacheDependency 对象组，当相关的任何缓存依赖发生变化时，缓存对象将过期，并从缓存中移除。

例如，创建一个 ASP.NET 应用程序，该应用程序可从 SQL Server 数据库和 XML 文件中输出数据集合。这时，可以创建一个与数据库关联的 SqlCacheDependency 对象和一个与 XML 文件相关联的 CacheDependency 对象，然后，使用 AggregateCacheDependency 类实现以上两个缓存依赖的聚合，最后，通过 Cache 类的 Insert 方法将聚合的缓存依赖添加到 Cache 对象中。下面是具体实现代码：

```
//自定义文件缓存依赖
CacheDependency dep1 = new CacheDependency(fileName);
//创建 SQL 数据缓存依赖，cmd 是一个 SqlCommand 实例
SqlCacheDependency dep2 = new SqlCacheDependency(cmd);
//添加到 CacheDependency 对象组中
CacheDependency[] deps = new CacheDependency[] { dep1, dep2 };
//调用 Add 方法，实现聚合缓存依赖
AggregateCacheDependency aggDep = new AggregateCacheDependency();
aggDep.Add(deps);
//调用 Insert 方法，添加应用程序缓存对象
Cache.Insert("key", DateTime.Now, aggDep);
```

5.5　应用程序缓存移除回调

缓存中的数据会因为各种原因而被移除，在某些情况下需要在缓存移除时将缓存中的数据转存到其他地方以防数据丢失，或是在缓存移除时做一些处理，这时就需要实现缓存移除回调。缓存移除回调就是对缓存的回调函数进行编码以实现既定的需求。

缓存移除回调委托的语法如下：

```
public delegate void CacheItemRemovedCallback(string key,
object value, CacheItemRemovedReason reason);
```

其中 key 表示要移除的缓存的键对象，value 就是缓存的值对象，reason 代表缓存移除的原因，它有以下 4 种可能：

- CacheItemRemovedReason.Removed——通过代码移除。
- CacheItemRemovedReason.Underused——内存不足，系统自动移除。
- CacheItemRemovedReason.Expired——缓存过期移除。
- CacheItemRemovedReason.DependencyChanged——依赖改变移除。

例如某考试系统为了防止客户端突然出现问题而导致客户已经填写的答案丢失，特在每个客户每完成一题就将该客户准考证号、题目编号和答案提交到服务器。由于可能出现大量的客户同时考试的情况，为加快服务器响应速度而使用缓存而不是数据库来临时存储客户提交到服务器中的信息。考试时长为两小时，因此设置缓存的过期时间为开考时间加两小时，则缓存过期时必须把缓存中的数据添加到数据库。主要代码如下：

```
using System.Web.Caching;
//添加到缓存，并同时设置移除回调
object obj = new object();
Cache.Add("examinfo", obj, null, DateTime.UtcNow.AddHours(2),
                System.Web.Caching.Cache.NoSlidingExpiration,
```



```
                    CacheItemPriority.Default, OnCacheRemoved);
//回调函数
void OnCacheRemoved(string key, object value, CacheItemRemovedReason reason)
{
      if (key == "examinfo" && (reason == CacheItemRemovedReason.Expired
            || reason == CacheItemRemovedReason.Underused))
      {
            //更新到数据库
      }
}
```

【小结】

- 缓存技术可以提高对页面的访问效率。
- 缓存分为页面输出缓存、页面部分缓存和应用程序数据缓存。
- 使用@OutputCache 指令配置页面缓存。
- 缓存依赖。
- 应用程序缓存是易失的，需要设置好缓存移除回调。

【自测题】

1. 缓存分为(　　)。

A. 页面输出缓存　　　　B. 相对缓存

C. 应用程序数据缓存　　D. 页面部分缓存

2. 设置页面缓存的指令是(　　)。

A. @Page　　B. @Control

C. @Assembly　　D. @OutputCache

3. Duration=3 代表缓存页面的时间是(　　)。

A. 3 小时　　B. 3 分钟

C. 3 秒钟　　D. 3 毫秒

4. 应用程序数据缓存类是下列哪一个？(　　)

A. Session　　B. Application　　C. Cookies　　D. Cache

5. CacheItemRemovedReason 枚举中表示依赖改变移除的是(　　)。

A. DependencyChanged　　B. Expired

C. Removed　　D. Underused

【上机部分】

上机目标

- 使用页面输出缓存
- 使用应用程序数据缓存

上机练习

◆ 第一阶段 ◆

练习 1：使用页面输出缓存。

【问题描述】

假设用户的站点包括一个从数据库表检索出来的显示产品信息的页面。在默认情况下，



每次用户访问产品页面时，都必须执行该页面并且从数据库检索数据。

【问题分析】

- 页面每次加载的时候都会从数据库中检索数据，这样开销很大，增加了服务器的压力。
- 启用页面输出缓存，这样页面就只执行一次，并且只从数据库检索一次数据。这样可以减轻 Web 应用程序和数据库服务器的负载。

【参考步骤】

(1) 新建一个网站 Emp_Cache。

(2) 添加一个名为 Duration1.aspx 的页面，要启用页面的输出缓存，在页面中添加如下页面处理指令即可。

```
<%@ OutputCache VaryByParam="None" Duration="3" %>
```

这条指令设置的页面输出被缓存 3 秒钟。

(3) 向 Duration1.aspx 添加如下代码:

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>缓存事例</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server"    Width="228px">
            <%#DateTime.Now.ToLongTimeString() %>
        </asp:Label>
```

```
        </div>
        </form>
</body>
</html>
```

(4) 编写后台代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
        //刷新时间显示，缓存 3 秒钟时间刷新一次
        this.DataBind();
}
```

在页面中的标签用来显示绑定时间，在加载页面后反复按 F5 键刷新页面，可以看到时间每三秒钟才改变一次。

练习 2：通过参数来多样化缓存内容。

【问题描述】

在练习 1 中，每次都是缓存整个页面的内容。但在实际开发中，可能是根据条件和参数的不同来有选择地缓存不同版本的页面内容。例如在 pubs 中，要想显示折扣信息，通常是要创建一个页面，其中能根据传递给页面的查询字符串参数来显示不同折扣的信息。

【问题分析】

- 根据商店 ID 查询不同的折扣信息，并生成不同的缓存页面。
- 配置@OutputCache 指令的 VaryByParam 属性，当设置 VaryByParam="none"时，所有传递给这个页面的任何查询字符串参数都被忽略。当设置 VaryByParam="*"时，表示只要给页面传递不同的参数，那么就创建一个该页面的不同缓存版本。
- 针对 ID 生成不同的页面缓存版本，应设置 VaryByParam="ID"。

【参考步骤】

(1) 向 Emp_Cache 添加一个名为 DiscountsByID.aspx 的页面。



(2) 启动页面缓存，设置按商店 ID 生成不同的缓存页面。

```
<%@ OutputCache Duration="30"  VaryByParam="ID"%>
```

(3) 在 DiscountsByID.aspx 页面添加如下设计代码:

```
<form id="form1" runat="server">
<div align="center">
    <h2>缓存时间：<%# DateTime.Now.ToLongTimeString() %></h2>
    <asp:GridView ID="gvData" runat="server" Width="338px" BackColor="White"
        BorderColor="#999999" BorderStyle="None" BorderWidth="1px" CellPadding="3"
        GridLines="Vertical">
        <FooterStyle BackColor="#CCCCCC" ForeColor="Black" />
        <RowStyle BackColor="#EEEEEE" ForeColor="Black" />
        <PagerStyle BackColor="#999999" ForeColor="Black" HorizontalAlign="Center" />
        <SelectedRowStyle BackColor="#008A8C" Font-Bold="True" />
        <HeaderStyle BackColor="#000084" Font-Bold="True" ForeColor="White" />
        <AlternatingRowStyle BackColor="#DCDCDC" />
    </asp:GridView>
</div>
</form>
```

(4) 在 DiscountsByID.aspx 的隐藏文件中添加如下代码:

```
protected void Page_Load(object sender, EventArgs e)
{
    //如果 ID 没有值，则默认为 1 号商品
    string strId = Request.QueryString["ID"];
    if (strId == null)
```

```
        strId = "1";

    string constr = "server=.;database=pubs;integrated security=true";
    //根据 ID 查询产品的详细信息
    SqlDataAdapter sda = new SqlDataAdapter(
                    "select * from discounts where stor_id=" + strId, constr);
    DataTable dt = new DataTable();
    sda.Fill(dt);
    //绑定 GridView 显示产品数据
    this.gvData.DataSource = dt;
    this.DataBind();
}
```

(5) 运行该页面，在 url 后改变 ID 参数的值，观察页面显示内容的变化。

先输入参数“id=1”时的结果如图 5-7 所示。

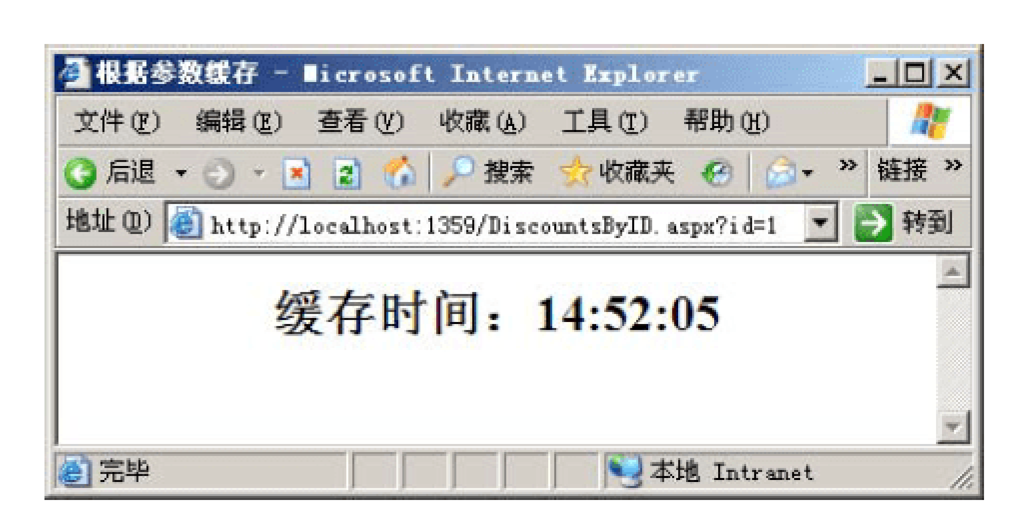


图 5-7

再输入参数“id=2”时的结果如图 5-8 所示。

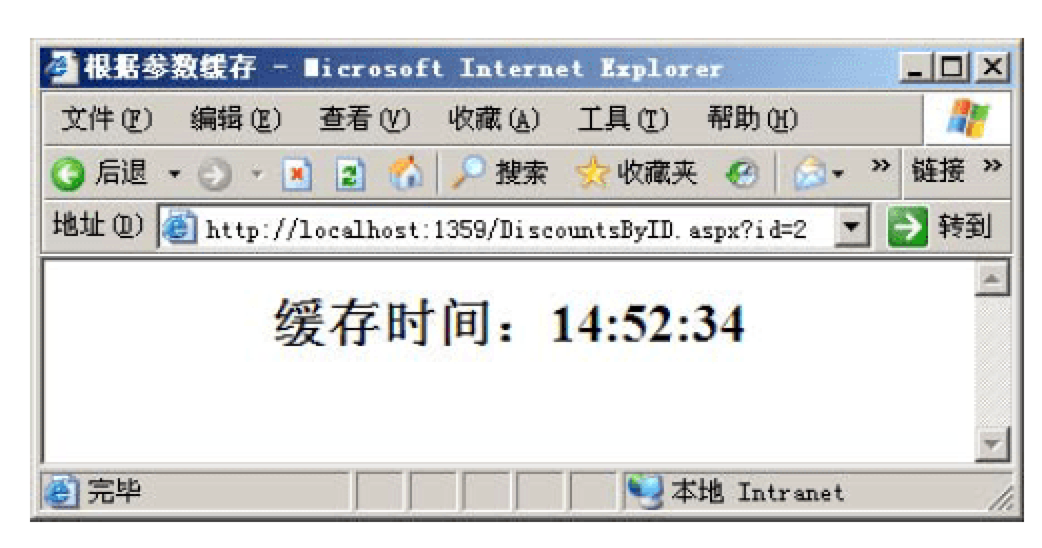


图 5-8



◆ 第二阶段 ◆

练习 1：应用程序数据缓存的绝对过期策略。

【问题描述】

我们知道，cookie 有过期时间的限制。应用程序数据缓存较 cookie 可以更加灵活、精确地设置过期时间。应用程序数据缓存的过期时间分为相对过期和绝对过期策略。请将上一题用应用程序数据缓存 Cache 来保存检索出来的折扣信息，设置缓存时间为绝对时间“2011-8-9 12:12:12”时过期。

练习 2：应用程序数据缓存的相对过期策略

【问题描述】

将上题缓存折扣信息的缓存时间改为相对时间 60 秒过期。试比较两种过期策略的不同点。

【课后作业】

1. 设计一个方法，判断用户是否禁用了浏览器的 cookie，如果禁用了则用 Cache 替代 cookie 来缓存信息。

2. 在理论课部分的学习中，通过示例 Example5_3 演示了如何在页面输出缓存中启用 SQL 数据缓存依赖。现将该示例中的数据缓存到应用程序数据缓存中，实现在应用程序数据缓存中启用 SQL 数据缓存依赖。

3. 比较 Application，Session 和 Cache 的优缺点。

第6章

母版页与站点导航



课程目标

- 了解母版页
- 使用母版页
- 使用 SiteMapPath 控件
- 使用 TreeView 控件
- 使用 Menu 控件





简　介

为了给访问者一致的感受，每个网站都需要具有统一的风格和布局。例如：整个网站具有相同的网页头尾、导航栏、功能条以及广告区等。对于这一点，在不同的技术发展阶段有着迥然不同的实现方法。在 ASP.NET 1.x 技术中，使用用户控件来实现网站的一致性。使用用户控件的方法，能够将网站中共用部分整合为一个控件。当需要实现时，只要在页面的适当位置引入该用户控件即可。

纵观以上方法，虽然都能够实现使网站一致性的要求，然而，其都不是开发技术的内建方法。也就是说，需要开发人员自己套用，才能够享用利用模板设计网页的好处。从 ASP.NET 2.0 开始给出一个新功能——母版页。设计母版页的目的，就是要从内部建立支持网页模板的功能，以实现网站一致性要求。本章内容将重点介绍在 ASP.NET 3.5 中母版页的使用，以便快速创建风格一致的应用程序。

6.1　母版页

6.1.1　母版页概述

从 ASP.NET 2.0 开始提供了母版页功能，它为提高工作效率、降低开发和维护强度，提供了有力支持。本节首先对母版页和内容页的概念，以及应用过程进行简单说明。然后，详细介绍母版页运行机制，以及母版页和内容页事件顺序，最后对母版页的优点做出总结。

1. 母版页基础知识

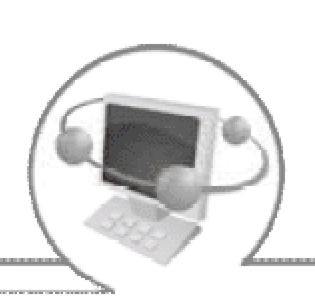
在 ASP.NET 中，可以将 Web 应用程序中的公用元素，例如，网站标志、广告条、导航条、版权声明等内容整合到母版页中。在这之前，开发人员更愿意将这一类似实现称为页面模板。实际上，可以将页面模板和母版页视为具有同一功能。换句话说，在 ASP.NET 中，可以将母版页看作是页面模板，而且是一种具有多项高级功能的页面模板。

同页面模板一样，母版页能够为 ASP.NET 应用程序创建统一的用户界面和样式，这是母版页的核心功能。在实现网站一致性的过程中，必须包含两种文件：一种是母版页，另一种是内容页。母版页扩展名是.master，是其封装页面中的公共元素。内容页实际上是普通的.aspx 文件，它包含除母版页之外的其他非公共内容。在运行过程中，ASP.NET 引擎将两种页面内容合并执行，最后将结果发给客户端浏览器。

常见母版页代码结构如下所示：

```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="MasterPage.master.cs"
Inherits="MasterPage" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <table>
          <tr>
            <td><asp:contentplaceholder id="Main" runat="server"/></td>
            <td><asp:contentplaceholder id="Footer" runat="server"/></td>
          </tr>
        </table>
    </form>
</body>
</html>
```

将以上母版页代码与普通.aspx 文件代码比较，可以发现，虽然两者存在一些相似之处，然而还是存在以下 3 点差异：一是母版页的扩展名是.master，所有以.master 为扩展名的文



件都是母版页，这一点与普通.aspx 文件不同。客户端浏览器可以向服务器发出请求，要求访问.aspx 文件，但是，如果请求的是母版页，则不能执行。客户端可以访问内容页，通过内容页对母版页的绑定，才能够间接访问母版页。二是普通.aspx 文件的代码头声明是<%@Page%>，而母版页文件的代码头声明与此不同，它必须声明为<%@Master%>。除此之外，母版页与普通.aspx文件在代码结构方面基本没有差异。例如，两者都需要声明<html>、<body>、<form>以及其他 Web 元素等。三是母版页中可以包括一个或者多个 ControlPlaceHolder 控件，而在普通.aspx 文件中是不包含该控件的。ContentPlaceHolder 控件起到一个占位符的作用，能够在母版页中标识出某个区域，该区域将被内容页中的特定代码代替。

以上介绍了母版页与.aspx 文件结构之间的差异。内容页和母版页联系紧密，下面简单介绍一下内容页。

内容页主要包含页面中的非公共内容。虽然内容页的扩展名是.aspx，但是，其代码结构与普通.aspx 文件差异很大。常见内容页的代码结构如下：

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.master" AutoEventWireup="true"
CodeFile="Index.aspx.cs"
Inherits="Index" Title="示例 11-1" %>

<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="Server">
…
</asp:Content>
```

由以上代码可知，内容页的代码主要分为两个部分：代码头声明和 Content 控件。内容页的代码头声明与普通.aspx 文件很相似。只是增加了属性 MasterPageFile 和 Title 设置。属性 MasterPageFile 用于设置该内容页所绑定的母版页的路径，属性 Title 用于设置页面 title 属性值。另外，在内容页中，还可以包括一个或者多个 Content 控件。页面中所有非公共内容都必须与 ContentPlaceHolder 控件相连接。通过以上设置，就可以实现母版页和内容

页的绑定。

ASP.NET 提供的母版页功能，允许开发人员创建真正意义上的页面模板。整个应用过程，可总结为“两个包含，一个结合”。

“两个包含”是指将页面内容分为非公共部分和公共部分，并且两者被分别包含在两个文件中。公共部分被包含在母版页中，非公共部分被包含在内容页中。开发人员可以根据所定义的公共内容，使用母版页来封装静态文本、HTML 元素以及 ASP.NET 服务器控件等多种 Web 元素。需要注意的是，即使公共内容处于页面中的不同位置，仍然可以使用母版页功能将其内容整合到一个母版页文件中。对于页面内容中的非公共部分，只需在母版页中使用一个或者多个 ContentPlaceHolder 控件来占位即可。ContentPlaceHolder 控件主要用于在母版页中，作为代替非公共部分的占位符出现，而具体内容则被放置在内容页中。内容页的创建相对简单，只需将非公共内容包含在不同的 Content 控件中即可。

“一个结合”是指通过控件应用以及属性设置等行为，将母版页和内容页有机结合。例如：母版页中 ContentPlaceHolder 控件的 ID 属性必须与内容页中 Content 控件的 ContentPlaceHolderID 属性绑定。

2．母版页运行机制

前文已经说明，单独的母版页是不能被用户所访问的。没有内容页支持，母版页仅仅是一个页面模板，没有更多的实用价值。同样道理，单独的内容页没有母版页支持，也不能够应用。由此可见，母版页与内容页关系密切，是不可分割的两个部分。只有同时正确创建和使用母版页以及内容页，才能发挥它们的强大功能。这一点，无论从代码结构，还是运行机制等方面都可以得到有力印证。

首先，从代码结构方面深入说明，以便用户了解母版页与内容页的运行机制。

由上文可知，母版页内容以页面公共部分为主，包括代码头、ContentPlaceHolder 控件以及其他常见 Web 元素。内容页主要包含页面非公共部分，包括两个部分：代码头和 Content 控件。Content 控件中包含着页面非公共内容。

在控件应用方面，母版页和内容页有着严格对应关系。母版页中包含多少个 ContentPlaceHolder 控件，那么内容页中也必须设置与其相同数目的 Content 控件。



在实际应用中，为了给整个网站创建一致的风格和样式，一个母版页可能被多个内容页绑定。只有正确处理母版页与内容页之间的控件对应关系，才能够准确、高效地创建 Web 应用程序。

下面重点对母版页的运行过程进行说明：

当客户端浏览器向服务器发出请求，要求浏览页面时，ASP.NET 执行引擎将执行内容页和母版页的代码，并将最终结果发送给客户端浏览器。

母版页和内容页的运行过程可以概括为以下 5 个步骤：

(1) 用户通过输入内容页的 URL 来请求某页。

(2) 获取内容页后，读取@Page 指令。如果该指令引用一个母版页，则也读取母版页。如果是第一次请求这两个页，则两个页都要进行编译。

(3) 母版页合并到内容页的控件树中。

(4) 各个 Content 控件的内容合并到母版页中相应的 ContentPlaceHolder 控件中。

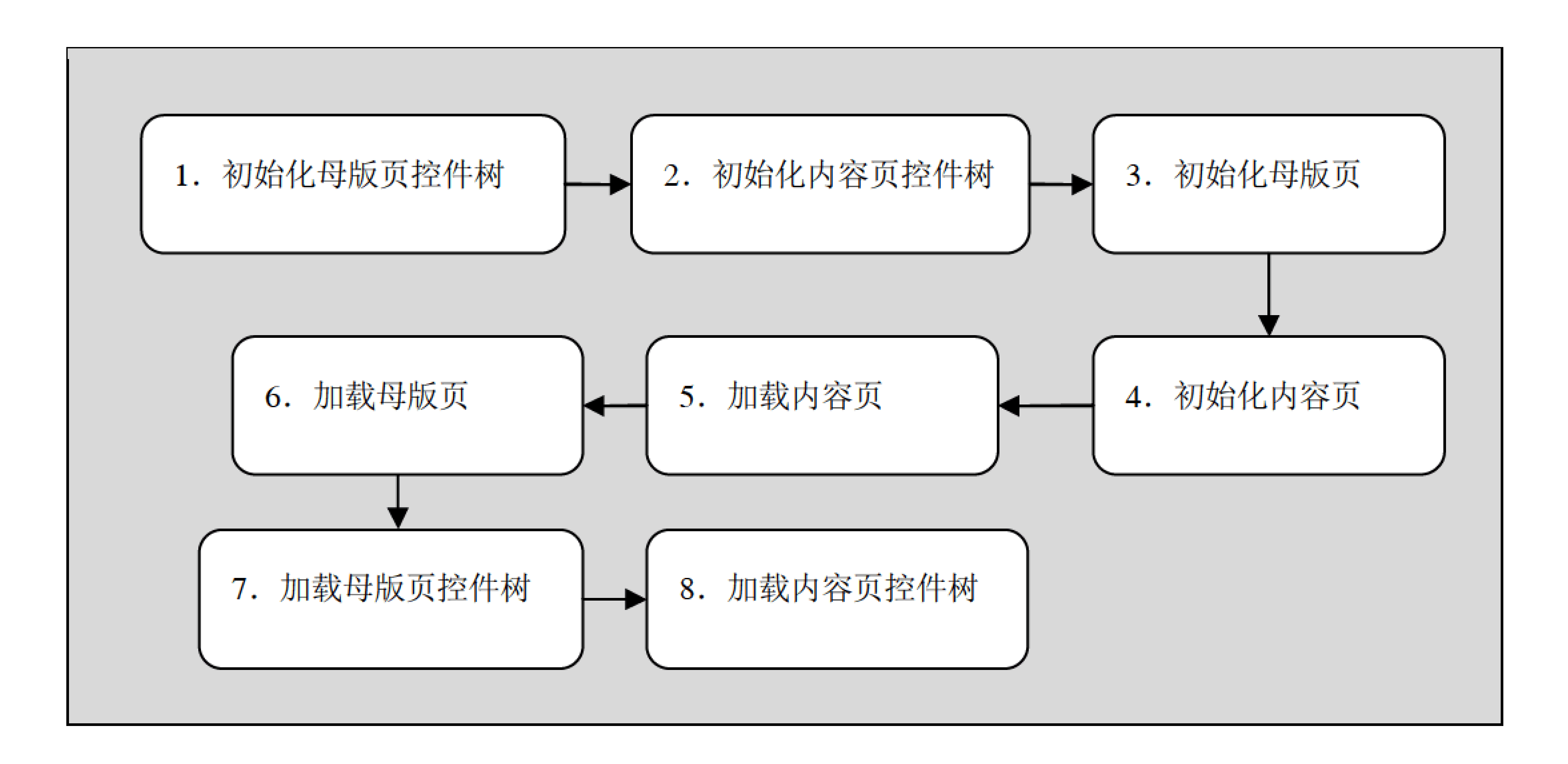
(5) 呈现得到的结果页。

整个过程具有很强的逻辑性，并且母版页和内容页配合得非常巧妙。从用户角度来看，合并后的母版页和内容页是一个完整的页面，并且其 URL 访问路径与内容页的路径相同。从开发人员角度来看，控件的巧妙应用和配合，是实现的关键。注意，在运行时，母版页成为内容页的一部分。实际上，母版页与用户控件的作用方式大致相同，即作为内容页的一个子级，并作为该页中的一个容器。

3. 母版页和内容页的事件顺序

通常情况下，母版页和内容页中的事件顺序对于页面开发人员并不重要。但是，如果所创建的事件处理程序取决于某些事件的可用性，那么了解母版页和内容页中的事件顺序就对用户很有帮助。本节将对母版面和内容页的事件顺序进行简要说明。

当访问结果页时，实际访问的是内容页和母版页。作为有着密切关系的两个页面，两者都要执行各自的初始化和加载等事件。具体过程如下所示：



加载母版页和内容页共需要经过 8 个过程。这 8 个过程显示初始化和加载母版页及内容页是一个相互交叠的过程。基本过程是，初始化母版页和内容页控件树；然后，初始化母版页和内容页页面；接着，加载母版页和内容页；最后，加载母版页和内容页控件树。以上 8 个过程对应着 11 个具体事件。这些事件如下所示：

- 母版页中控件的 Init 事件。
- 内容页中 Content 控件 Init 事件。
- 母版页 Init 事件。
- 内容页 Init 事件。
- 内容页 Load 事件。
- 母版页 Load 事件。
- 内容页中 Content 控件 Load 事件。
- 内容页 PreRender 事件。
- 母版页 PreRender 事件。
- 母版页控件 PreRender 事件。
- 内容页中 Content 控件 PreRender 事件。

实际上，8 个过程或者是 11 个事件都用于说明母版页和内容页中的具体事件顺序。内容页和母版页中引发相同的事件。例如：两者都引发 Init、Load 和 PreRender 事件。引发



事件的一般规律是，初始化 Init 事件从最里面的控件(母版页)向最外面的控件(Content 控件及内容页)引发，所有其他事件则从最外面的控件向最里面的控件引发。需要牢记，母版页会合并到内容页中，并被视为内容页中的一个控件，这一点十分有用。

在创建应用程序时，必须注意以上事件顺序。例如，当在内容页中访问母版页的属性或者服务器控件时，如果按照过去的处理思路，可能会在内容页的 Page_Load 事件处理程序中加以实现。由前文可知，在母版页 Load 事件引发之前，内容页 Load 事件已经引发，那么过去的思路显然是不正确的。

4. 母版页的优点

使用母版页功能，可以为 ASP.NET 应用程序页面创建一个通用的外观。开发人员可以利用母版页功能创建一个单页布局，然后将其应用到多个内容页中。总结起来，母版页具有以下 4 个优点。

- 有利于站点修改和维护，降低开发人员的工作强度。
- 提供高效的内容整合能力。
- 有利于实现页面布局。
- 提供一种便于利用的对象模型。

6.1.2　创建母版页

虽然母版页和内容页功能强大，但是其创建和应用过程并不复杂。本节和下一节将使用 Visual Studio 2008 创建母版页和内容页。

母版页中包含的是页面公共部分，即网页模板。因此，在创建之前，必须判断哪些内容是页面公共部分，这就需要从分析页面结构开始。例如要创建一个如图 6-1 所示的页面，将该页面命名为 Index.aspx。通过分析可知，该页面的结构如图 6-2 所示。

页面 Index.aspx 由 4 个部分组成：页头、页尾、内容 1 和内容 2。其中页头和页尾是 Index.aspx 所在网站中页面的公共部分，网站中许多页面都包含相同的页头和页尾。内容 1 和内容 2 是页面的非公共部分，是 Index.aspx 页面所独有的，结合母版页和内容页的有关知识可知，如果使用母版页和内容页来创建页面 Index.aspx，那么必须创建一个母版页

MasterPage.master 和一个内容页 Index.aspx。其中母版页包含页头和页尾等内容，内容页中则包含内容 1 和内容 2。

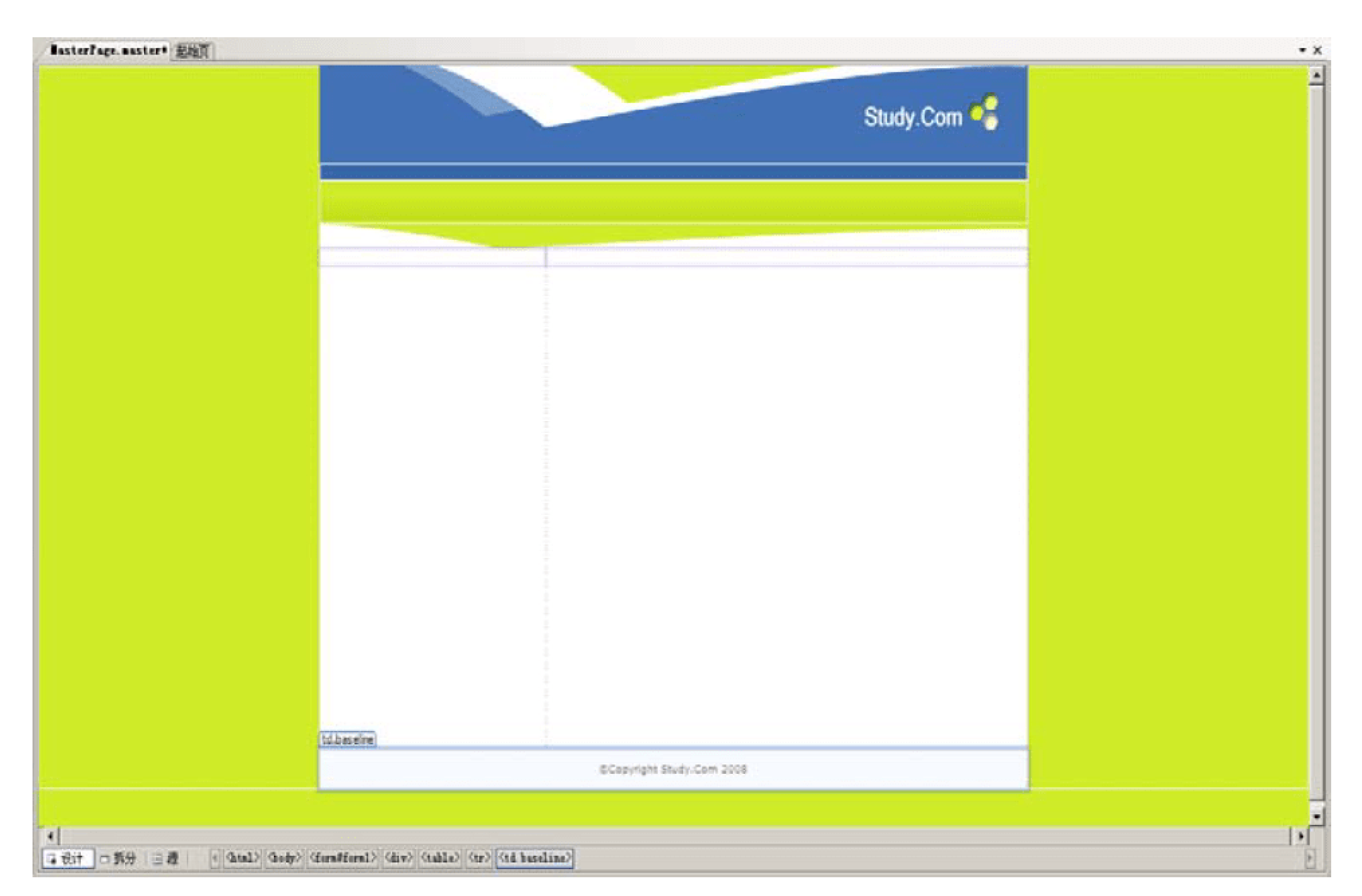


图 6-1

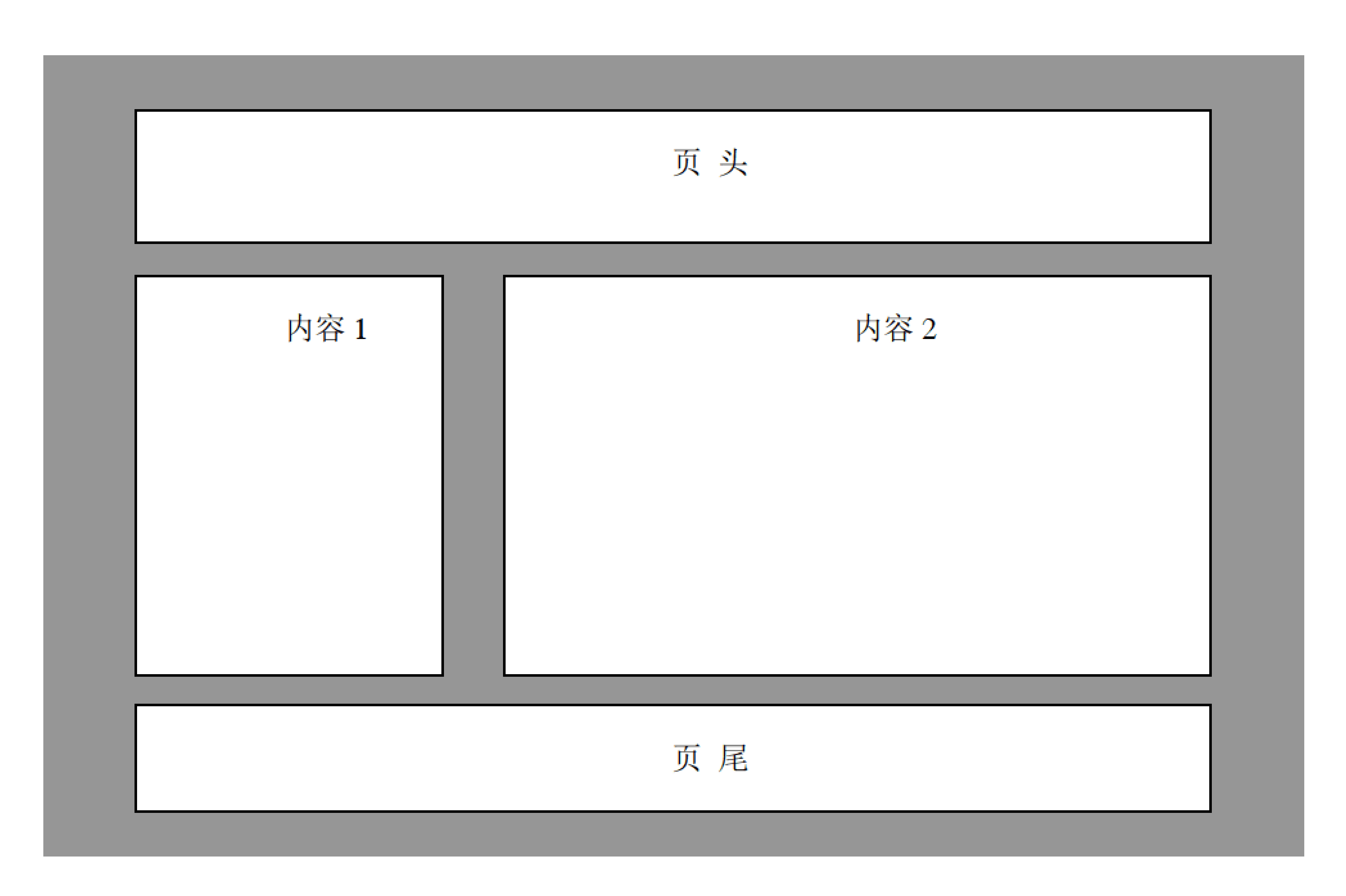


图 6-2



使用 Visual Studio 2008 创建一个普通 Web 站点，然后，在站点根目录下创建一个名为 MasterPage.master 的母版页。由于这是一个添加新文件的过程，因此，选择“网站”命令菜单中的“添加新项”选项，可以打开如图 6-3 所示的对话框。



图 6-3

由于此例创建的是母版页，因此，需要选择母版页图档，并且设置文件名为 MasterPage.master。需要注意的是，该窗口中还有一个复选框“将代码放在单独的文件中”。默认情况下，该复选框处于选中状态。表示 Visual Studio 2008 将会为 MasterPage.master 文件应用代码隐藏模型，即在创建 MasterPage.master 文件的基础上，自动创建一个与该文件相关的 MasterPage.master.cs 文件。如果不选中该复选框，那么只会创建一个 MasterPage.master 文件而已。

在创建 MasterPage.master 文件之后，接着就可以开始编辑该文件了。根据前文说明，母版页中只包含页面公共部分，因此，MasterPage.master 中主要包含的是页头和页尾的代码。具体代码如下所示：

```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="MasterPage.master.cs"
Inherits="MasterPage" %>

<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head id="Head1" runat="server">
    <title></title>
    <link href="css/myfreetemplates.css" rel="stylesheet" type="text/css" />
</head>
<body background="images/pixi_lime.gif" leftmargin="0" topmargin="0">
    <form id="form1" runat="server">
        <div align="center">
            <table width="763" height="100%" border="0" cellpadding="0"
                 cellspacing="0" bgcolor="#FFFFFF">
                <tr>
                    <td width="763" height="86" align="right" valign="top">
                        <img src="images/topic.gif"></td>
                </tr>
                <tr>
                    <td width="763" height="53" align="right" valign="bottom"
                       background="images/nav_bg.gif">
                    </td>
                </tr>
                <tr>
                    <td width="763" height="22" align="right" valign="top">
                    <img src="images/toppic2.gif" width="763" height="22"></td>
                </tr>
                <tr>
                    <td width="763" valign="top">
                        <table width="100%" border="0" cellspacing="0"
                              cellpadding="0">
```



```
                                        <tr>
                                            <td width="244" valign="top">
<asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
</asp:ContentPlaceHolder>
                                            </td>
                                            <td valign="top" align="left">
<asp:ContentPlaceHolder ID="ContentPlaceHolder2" runat="server">
</asp:ContentPlaceHolder>
                                            </td>
                                        </tr>
                                    </table>
                                </td>
                            </tr>
                            <tr>
                                <td width="763" height="1"
                                    background="images/pixi_lime.gif">
                                    <img src="images/pixi_lime.gif" width="1" height="1">
</td>
                            </tr>
                            <tr>
                                <td width="763" height="35" align="center" class="baseline">
                                    &copy;Copyright Study.Com 2008</td>
                            </tr>
                        </table>
                    </div>
    </form>
```

```
</body>
</html>
```

以上是母版页 MasterPage.master 的源代码，与普通的.aspx 源代码非常相似。例如，包括<html>、<body>、<form>等 Web 元素，但是，与普通页面还是存在差异。差异主要有两处(粗体代码所示)：差异一是代码头不同，母版页使用的是 Master，而普通.aspx 文件使用的是 Page。除此之外，两者在代码头方面是相同的；差异二是母版页中声明了控件 ContentPlaceHolder，而在普通.aspx 文件中是不允许使用该控件的。在 MasterPage.master 的代码中，共声明了两个 ContentPlaceHolder 控件，用于在页面模板中为内容 1 和内容 2 占位。ContentPlaceHolder 控件本身并不包含具体内容设置，仅是一个控件声明。

图 6-4 所示，显示了 MasterPage.master 文件的设计时视图。

使用 Visual Studio 2008 可以对母版页进行编辑，并且它完全支持“所见即所得”功能。无论在代码模式下，还是设计模式下，使用 Visual Studio 2008 编辑母版页的方法，与编辑普通.aspx 文件是相同的。图中两个矩形框表示 ContentPlaceHolder 控件。开发人员可以直接在矩形框中添加内容，所设置内容的代码将包含在 ContentPlaceHolder 控件声明代码中。

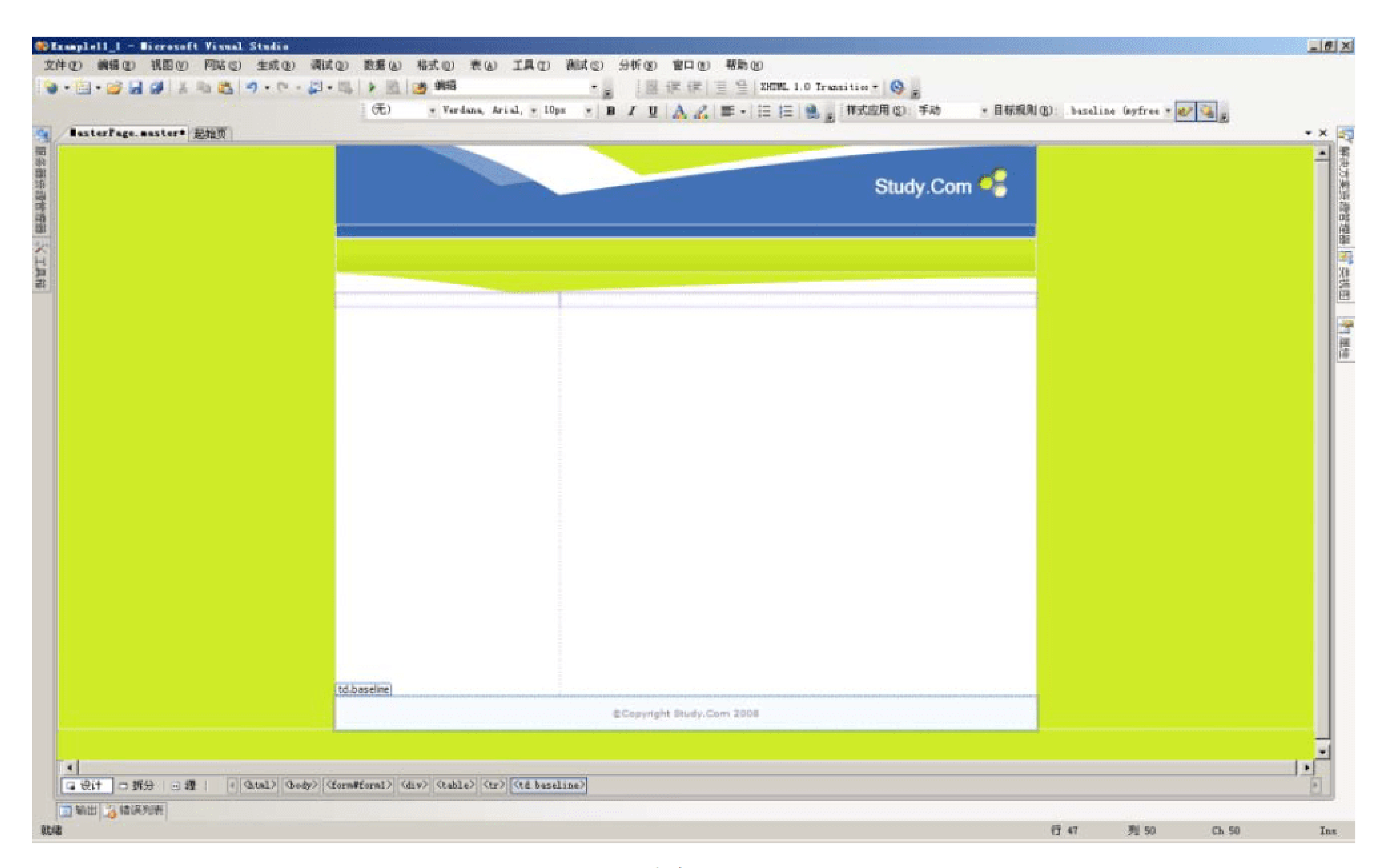


图 6-4



6.1.3　创建内容页

在创建一个完整的母版页之后，接下来必然要创建内容页。从用户访问的角度来讲，内容页与最终结果页的访问路径相同，这好像表明两者是同一文件，实际不然。结果页是一个虚拟的页面，没有实际代码，其代码内容是在运行时状态下母版页合并的结果。在开始介绍内容页之前，还有两个概念要强调：一是内容页中所有内容必须包含在 Content 控件中；二是内容页必须绑定母版页。虽然内容页的扩展名与普通 ASP.NET 页面相同，但是，其代码结构有着很大差别。在创建内容页的过程中，必须时刻牢记以上两个概念。

与创建母版页差不多，创建内容页的过程比较简单。选择“网站”命令菜单中的“添加新项”命令，或者在解决方案管理器中右击项目，在弹出的快捷菜单中选择“添加新项”命令，就可以打开如图 6-5 所示的对话框。



图 6-5

图 6-5 所示，要求选择新建文件类型。由于内容页与普通.aspx 页面的扩展名相同，因此，选择的是 Web 窗体图标。接着，还需要设置文件名 Index.aspx。设置完成之后，不可直接单击“添加”按钮。因为，内容页必须绑定母版页，所以还需要选中复选框“将代码放在单独的文件中”和“选择母版页”。前者在前文中已经说明，重点是说明后者。“选择母版页”复选框用于设置所创建的 Web 窗体是否绑定母版页。如果创建的是内容页，那么必须选中该复选框。结束以上操作之后，可以单击“确定”按钮，从而弹出如图 6-6 所示的对话框。

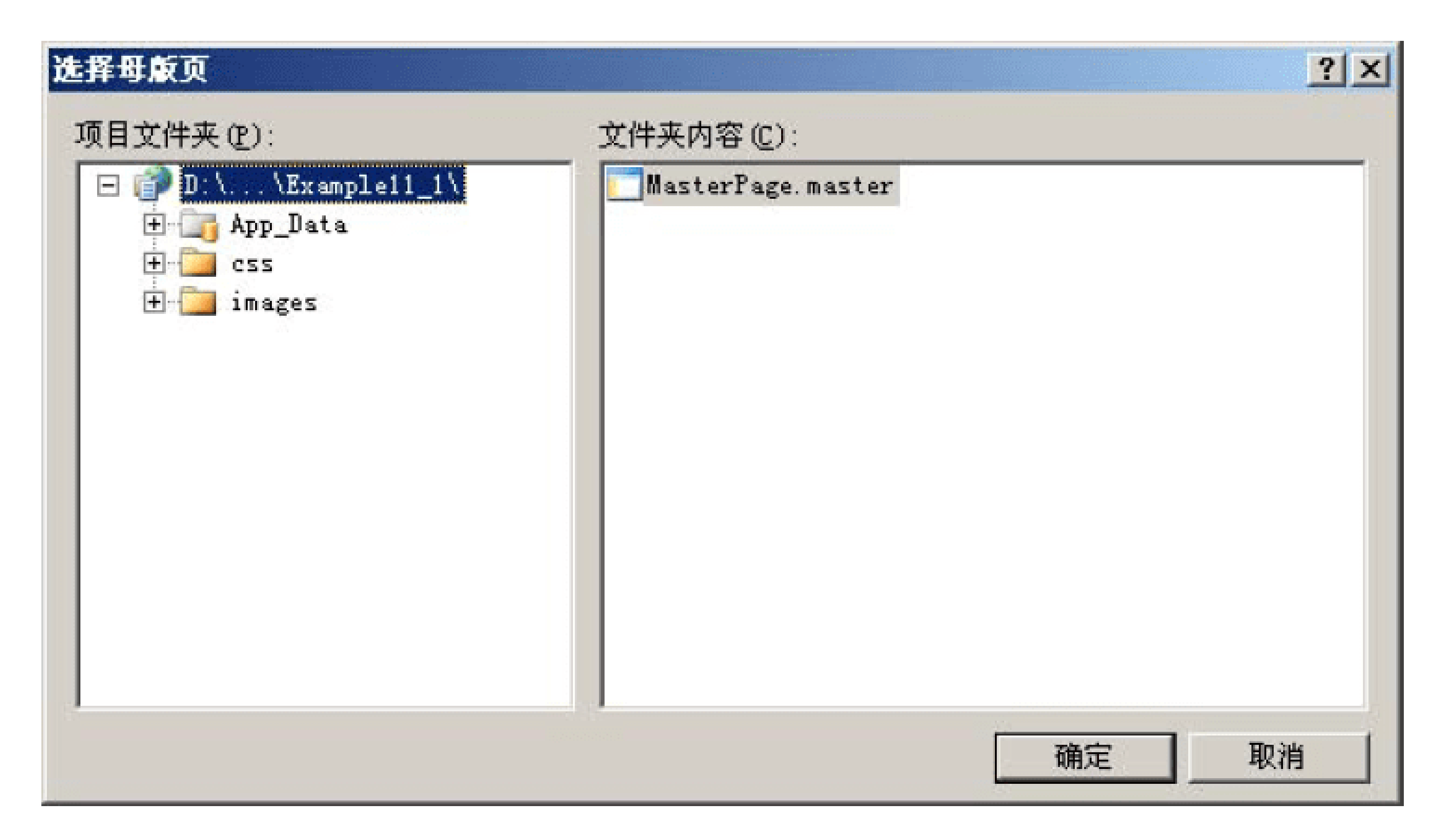


图 6-6

对话框左侧是项目文件，右侧是文件夹中的母版页列表，图 6-6 所示对话框中已经列举了刚刚创建的母版页 MasterPage.master，选中该文件，单击“确定”按钮即可。经过以上步骤，就顺利创建了一个绑定母版页 MasterPage.master 的内容页 Index.aspx 了。

下面列出了内容页 Index.aspx 的源代码：

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.master" AutoEventWireup="true"
CodeFile="Index. aspx.cs"
Inherits="Index" Title="示例" %>

<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="Server">
    <img src="images/pagepic.gif" width="244" height="223" />
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder2" runat="Server">
    <p> </p>
    <p> </p>
    <h1>网站介绍</h1>
<p>本页面采用来自 ASP.NET 3.5 技术的母版页新特性进行开发。<p>
    <br>
</asp:Content>
```



以上粗体代码是内容页的代码结构。从中可以看出，内容页与普通.aspx 文件在代码上的不同。内容页没有<html>、<body>、<form>等关键 Web 元素，这些元素都放置在母版页中。内容页中除了代码声明，仅包含 Content 控件。内容页的代码头声明与普通.aspx 文件相似。但是，新增加了两个属性 MasterPageFile 和 Title。属性 MasterPageFile 用于设置该内容页所绑定的母版页的路径，属性 Title 用于设置页面 title 值。在创建内容页的过程中，由于已经指定了所绑定母版页，因此，Visual Studio 2008 将自动设置 MasterPageFile 的属性值。另外，在源代码中，还设置了两个 Content 控件 Content1 和 Content2。两个控件内部包含的内容是页面的非公共部分。通过设置属性 ContentPlaceHolderID，将 Content1 与母版页的 ContentPlaceHolder1 对应，将 Content2 与母版页的 ContentPlaceHolder2 对应。在页面运行时，Content 控件中包含的内容将显示在母版页中的对应位置。



图 6-7 所示显示了内容页 Index.aspx 的设计视图。

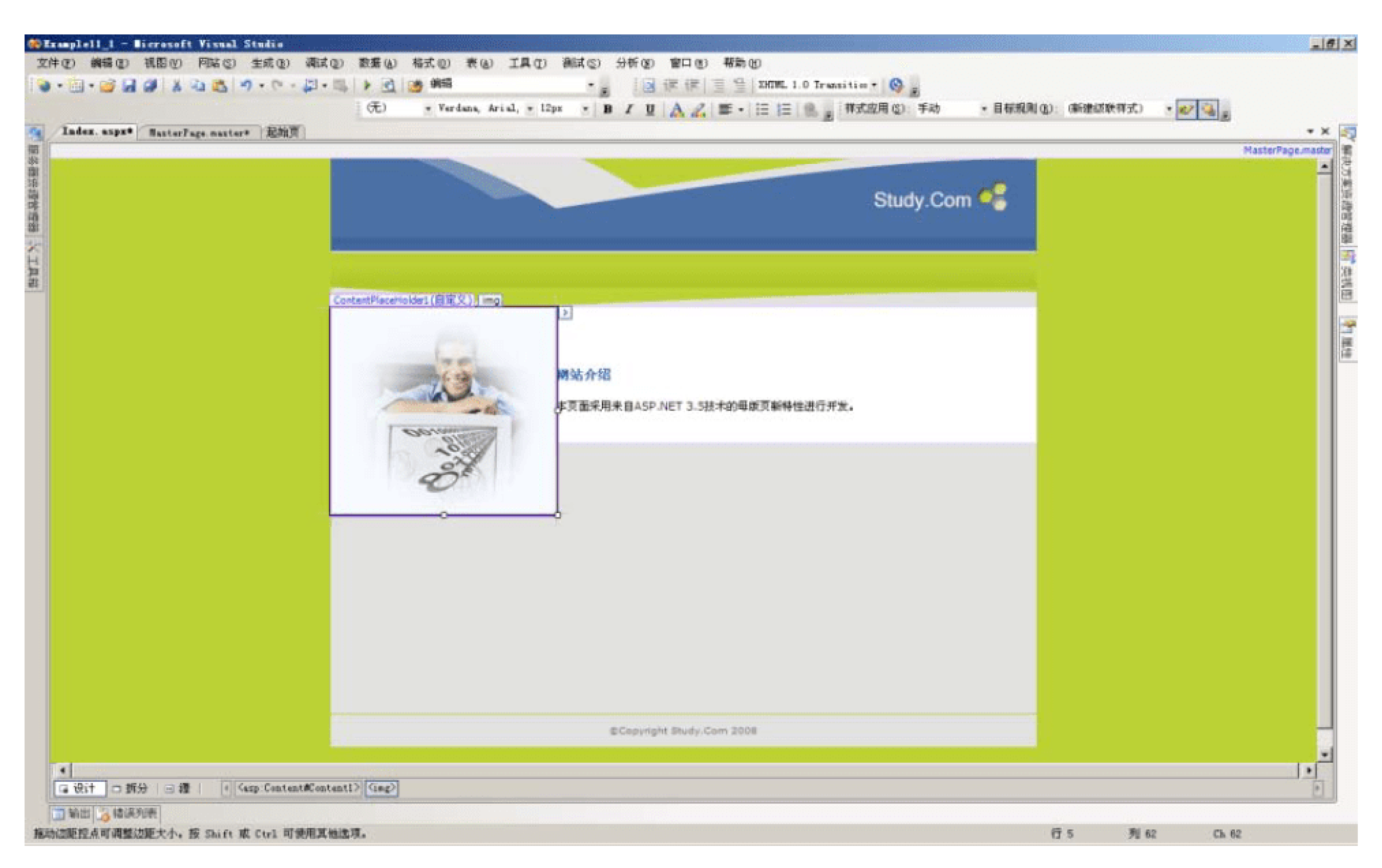
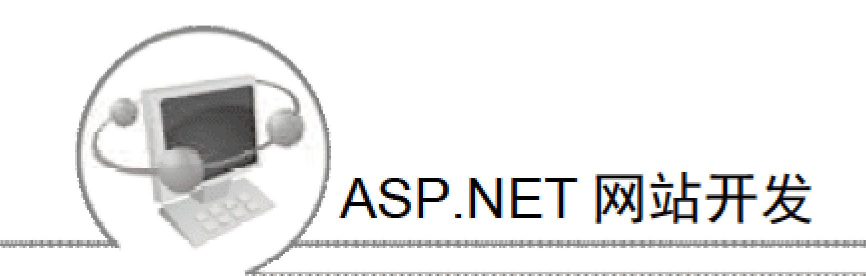


图 6-7

同母版页一样，Visual Studio 2008 支持对于内容页的可视化编辑，并且这种支持是建立在只读显示母版页内容基础上的。在编辑状态下，可以查看母版页和内容页组合后的页面外观，但是，母版页内容是只读的(呈现灰色部分)，不可被编辑，而内容页则可以进行编辑。如果需要修改母版页内容，则必须打开母版页。



6.1.4　访问母版页

母版页和内容页都可以包含控件的事件处理程序。对于控件而言，事件是在本地处理的，即内容页中的控件在内容页中引发事件，母版页中的控件在母版页中引发事件。控件事件不会从内容页发送到母版页。同样，也不能在内容页中处理来自母版页控件的事件。如果需要访问母版页的控件和属性，将是非常困难的。本节重点介绍访问母版页控件和属性的方法，内容包括：使用 FindControl，获取母版页控件的实现方法，使用 MasterType 指令，获取母版页控件引用的实现方法，使用 MasterType 指令，访问母版页简单自定义属性的实现方法。

1. 使用 FindControl 方法获取母版页控件引用

如果需要从母版页内部获取控件属性值，是非常简单的。只需通过 Page 对象获取控件引用，继而访问控件属性值即可。如果需要从内容页获取母版页控件引用，可能会变得有些困难。本节重点介绍使用 FindControl 方法，从内容页获取母版页控件引用的方法。

在内容页中，核心对象 Page 具有一个公共属性 Master，该属性能够实现对相关母版页基类 MasterPage 的引用。隶属于母版页的 MasterPage 相当于普通 ASP.NET 页面的 Page 对象。由此可以使用 MasterPage 对象，实现对于母版页中各子对象的访问。由于母版页中的控件是受保护的，因此不能被直接访问，必须使用 MasterPage 的 FindControl 方法实现。本节将实现一个使用 FindControl 的方法，获取母版页中控件的引用的示例。如图 6-8 所示是示例的效果。

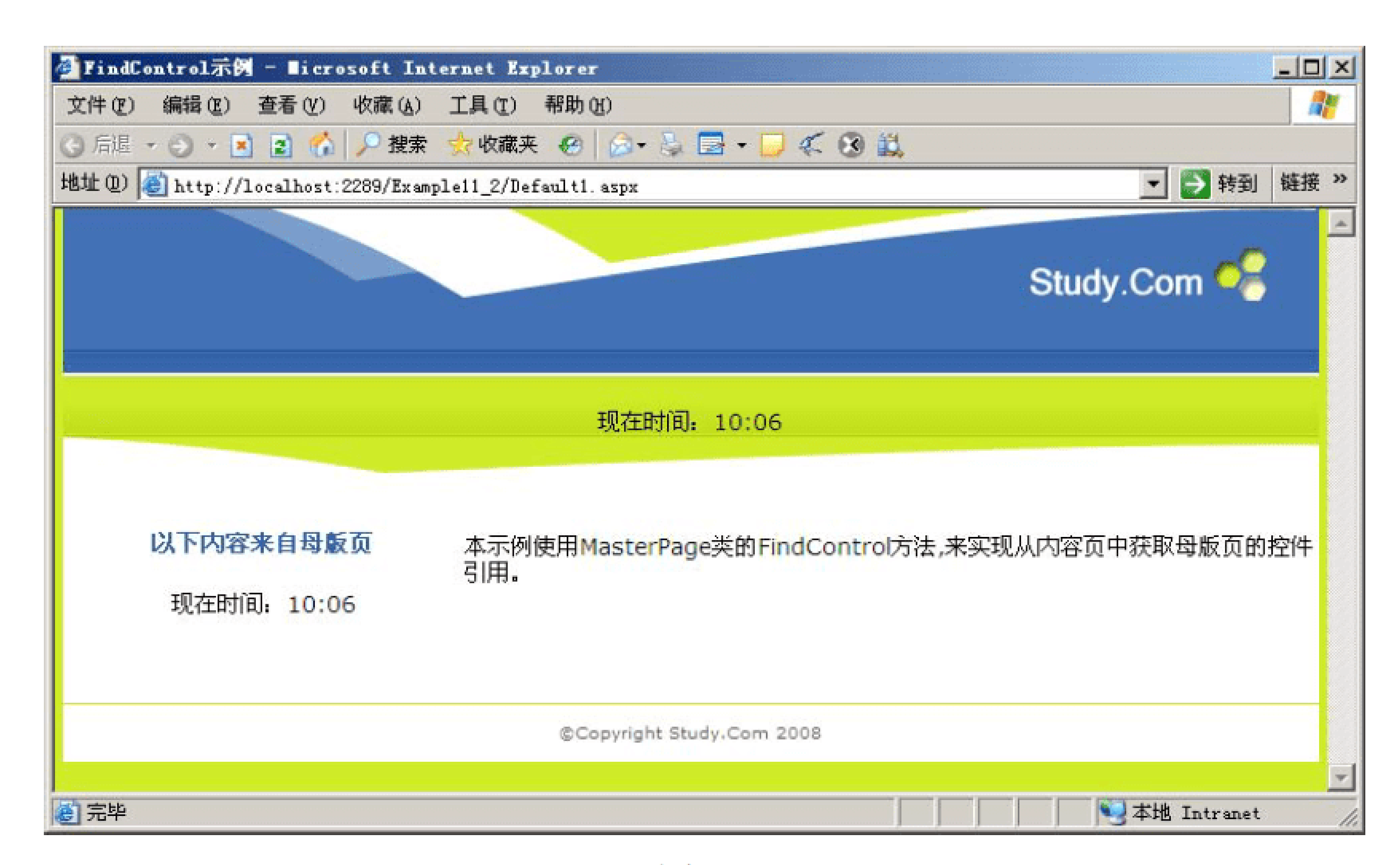


图 6-8



该 Web 页面由母版页和内容页组成。母版页包括页头和页尾，内容页则包含中间的非公共部分。在母版页中定义一个 Label 控件，使其显示当前时间，然后，在内容页中编写代码，实现对母版页中 Label 控件的引用，进而访问其 Text 属性值，并将访问结果显示在内容页中。

母版页 MasterPager1.master 源代码如下：

```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="MasterPage1.master.cs"
    Inherits="MasterPage1" %>

<script runat="server">
    void Page_Load(Object sender, EventArgs e)
    {
        LabelInMaster.Text = "现在时间：" +
                                         System.DateTime.Now.ToShortTimeString();
    }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title></title>
    <link href="css/myfreetemplates.css" rel="stylesheet" type="text/css">
</head>
<body background="images/pixi_lime.gif" leftmargin="0" topmargin="0">
    <form id="form1" runat="server">
        <div align="center">
            <table width="763" height="100%" border="0" cellpadding="0"
                  cellspacing="0" bgcolor="#FFFFFF">
```

```
                <tr>
                    <td width="763" height="86" align="right" valign="top">
                        <img src="images/topic.gif"></td>
                </tr>
                <tr>
                    <td width="763" height="53" align="center" valign="bottom"
                            background="images/nav_bg.gif">
                    <asp:Label ID="LabelInMaster" runat="server"></asp:Label>
                    </td>
                </tr>
                <tr>
                    <td width="763" height="22" align="right" valign="top">
                        <img src="images/toppic2.gif" width="763" height="22">
</td>
                </tr>
                <tr>
                    <td width="763" valign="top">
                        <table width="100%" border="0" cellspacing="0"
                              cellpadding="0">
                            <tr>
                                <td width="244" valign="top">
<asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
</asp:ContentPlaceHolder>
                                </td>
                                <td valign="top" align="left">
<asp:ContentPlaceHolder ID="ContentPlaceHolder2" runat="server">
```



```
</asp:ContentPlaceHolder>
                                        </td>
                                   </tr>
                              </table>
                         </td>
                    </tr>
                    <tr>
                         <td width="763" height="1"
                           background="images/pixi_lime.gif">
                              <img src="images/pixi_lime.gif" width="1" height="1">
</td>
                    </tr>
                    <tr>
                         <td width="763" height="35" align="center" class="baseline">
                              &copy;Copyright Study.Com 2008</td>
                    </tr>
               </table>
          </div>
     </form>
</body>
</html>
```

以上粗体代码部分是需要注意的。在粗体代码中，定义了一个控件 LabelInMaster，并且通过 Page_Load 事件处理程序，使该控件显示当前时间。这里的 LabelInMaster 就是随后在内容页中被引用的控件。

下面列举了内容页 Default.aspx 文件代码：

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage1.master" AutoEventWireup="true"
CodeFile="Default1.aspx.cs"
Inherits="Default1" Title="示例" %>

<script runat="server">
    void Page_LoadComplete(Object sender, EventArgs e)
    {
     LabelInContent.Text = (Master.FindControl("LabelInMaster") as Label).Text;
    }
</script>

<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
    <p>
         </p>
    <h1 align="center">
        以下内容来自母版页</h1>
    <asp:Label ID="LabelInContent" runat="server"></asp:Label>
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder2" runat="server">
    <br />
    <p>
本示例使用 MasterPage 类的 FindControl 方法，来实现从内容页中获取母版页的控件引用。
    </p>
    <br />
</asp:Content>
```

以上内容页代码粗体部分是本例重点。其核心在于定义 Page_LoadComplete 事件处理



程序及使用 FindControl 方法。由前文内容可知，在母版页 Page_Load 事件引发之前，内容页 Page_Load 事件已经引发。在这种情况下，从内容页中访问母版页控件引用不大容易。幸运的是，ASP.NET 创建了一个新事件 LoadComplete。开发人员可以在该事件对应的事件处理程序 Page_LoadComplete 中实现对母版页控件的引用。在本例的 Page_LoadComplete 实现中，通过 Master.FindControl 方法获取了母版页中控件 LabelInMaster 的属性值，然后，将该值赋予内容页中 LabelInContent 控件的 Text 属性，通过以上步骤，即可完成从内容页访问母版页控件引用的任务。

2. 使用 MasterType 指令获取母版页控件引用

上节对使用 FindControl 方法访问母版页控件引用进行了说明。通过示例可以看出，实现方法简单易行，能够解决很多实际中遇到的困难。本节将通过一个简单示例，说明另一种获取母版页控件引用的方法，其核心是应用 MasterType 指令。

在内容页中使用 MasterType 指令后，将使得内容页中的 Master 属性被强类型化。也就是说，通过 MasterType 指令，可以创建与内容页相关的母版页的强类型引用。由此，可以在内容页中，使用 Master 对象访问母版页的公共方法、属性和控件等成员。另外，在设置 MasterType 指令时，还必须设置 VirtualPath 属性以便指定与内容页相关的母版页存储地址。下面列举了一个示例，通过 MasterType 指令，实现了从内容页获取对母版页控件的引用。示例效果如图 6-9 所示。

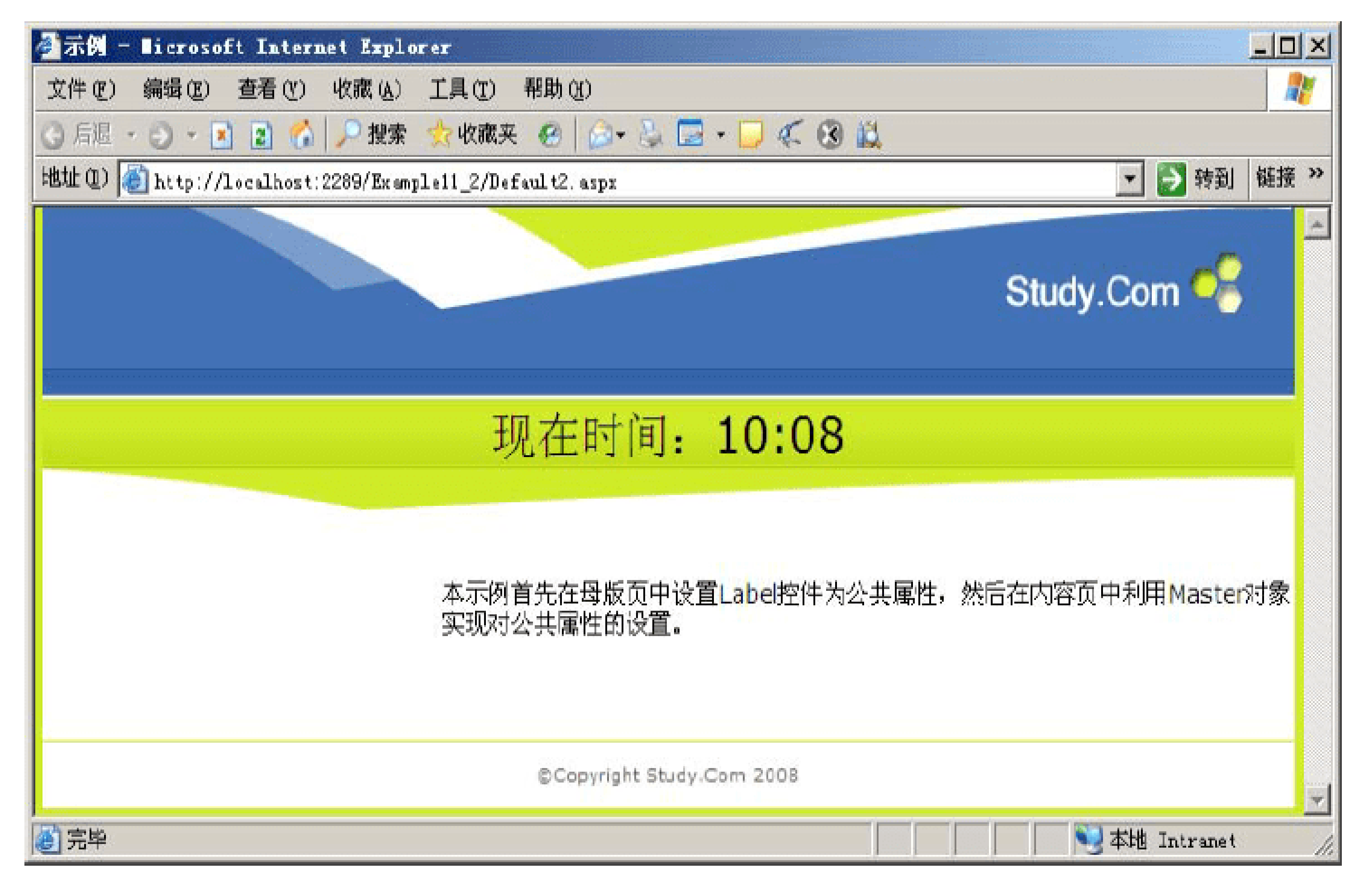


图 6-9

本例在母版页中实现一个自定义公共属性，该属性引用母版页内的 Label 控件。通过内容页中的 MasterType 指令，对母版页实施强类型化。由此，实现了从内容页访问母版页中 Label 控件的引用。

下面是母版页的代码：

```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="MasterPage2.master.cs"
    Inherits="MasterPage2" %>

<script runat="server">
    public Label MasterPageLabel
    {
        get
        {
            return Label1;
        }
        set
        {
            Label1 = value;
        }
    }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title></title>
    <link href="css/myfreetemplates.css" rel="stylesheet" type="text/css">
</head>
```



```
<body background="images/pixi_lime.gif" leftmargin="0" topmargin="0">
    <form id="form1" runat="server">
        <div align="center">
            <table width="763" height="100%" border="0" cellpadding="0"
                      cellspacing="0" bgcolor="#FFFFFF">
                <tr>
                    <td width="763" height="86" align="right" valign="top">
                        <img src="images/topic.gif"></td>
                </tr>
                <tr>
                    <td width="763" height="53" align="center" valign="bottom"
                            background="images/nav_bg.gif">
                        <asp:Label ID="Label1" runat="server"></asp:Label>
                    </td>
                </tr>
                <tr>
                    <td width="763" height="22" align="right" valign="top">
                        <img src="images/toppic2.gif" width="763" height="22">
</td>
                </tr>
                <tr>
                    <td width="763" valign="top">
                        <table width="100%" border="0" cellspacing="0"
                                cellpadding="0">
                            <tr>
                                <td width="244" valign="top">
```

```
<asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
</asp:ContentPlaceHolder>
                                        </td>
                                        <td valign="top" align="left">
<asp:ContentPlaceHolder ID="ContentPlaceHolder2" runat="server">
</asp:ContentPlaceHolder>
                                        </td>
                                    </tr>
                                </table>
                            </td>
                        </tr>
                        <tr>
                            <td width="763" height="1"
background="images/pixi_lime.gif">
                                <img src="images/pixi_lime.gif" width="1" height="1">
</td>
                        </tr>
                        <tr>
                            <td width="763" height="35" align="center" class="baseline">
                                &copy;Copyright Study.Com 2008</td>
                        </tr>
                    </table>
                </div>
            </form>
</body>
</html>
```



如上代码所示，母版页中实现了一个自定义公共属性 MasterPageLabel，该属性引用了页面中的 Label1 控件。无论何处对 MasterPageLabel 属性的设置和访问，实际都是在对 Label1 控件进行操作。然而，如果没有通过属性 MasterPageLabel 公开 Label1 控件，那么很难从母版页以外访问 Label1 控件。因为 Label1 控件是一个非公共成员。如果能够从外部引用母版页，那么意味着可以访问 MasterPageLabel 属性，进而访问 Label1 控件。下面列出了内容页源代码：

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage2.master" AutoEventWireup="true"
CodeFile="Default2.aspx.cs"
Inherits="Default2" Title="示例" %>

<%@ MasterType VirtualPath="~/MasterPage2.master" %>

<script runat="server">
    void Page_Load(Object sender, EventArgs e)
    {
        Master.MasterPageLabel.Text = "现在时间：" +
                                    System.DateTime.Now.ToShortTimeString();
        Master.MasterPageLabel.Font.Size = 20;
    }
</script>

<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
    <p> </p>
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder2" runat="server">
    <br />
```

```
    <br />
    <p>
        本示例首先在母版页中设置 Label 控件为公共属性，然后在内容页中利用 Master 对象实现
        对公共属性的设置。</p>
    <br />
    <br />
</asp:Content>
```

如上代码所示，在内容页代码头的设置中，除添加了常见的<%@ Page %>指令，还新增了一个<%@ MasterType %>指令，并在其中设置了 VirtualPath 属性。另外，实现了 Page_Load 事件处理程序，其通过 Master 对象的 MasterPageLabel 属性，来设置母版页中的 Lable1 控件。也许大家会有疑问，在<%@ Page %>指令中，已经通过 MasterPageFile 属性指定了母版页 URL 地址，而在<%@ MasterType %>指令中，却又通过 VirtualPath 属性设置母版页 URL 地址，两者是否可以只取其一呢？这是不允许的。因为，MasterPageFile 属性用于设置内容页绑定的母版页 URL 地址，而 VirtualPath 属性用于设置被强类型化的母版页 URL 地址。常见情况下，两个属性值相同，但是仍然必须同时正确设置 PmaterPageFile 和 VirtualPath 属性。

3. 访问母版页简单自定义属性

母版页中的自定义属性主要包括两种：一种是简单自定义属性，其属性值可以很容易转换为字符串表达式，这种属性的类型通常为 Boolean、Byte、Char、Double、Enum、Int32、DateTime 等简单数值类型，以及 String 类型和枚举类型等。另一种是复杂自定义属性，其属性值多是控件、类等类型，例如，上一节中定义的 MasterPageLabel 就是一个复杂自定义属性。本节将通过一个示例说明访问母版页简单自定义属性的实现方法。如图 6-10 所示显示了示例效果图。

同上一示例类型，本例同样在母版页中显示当前时间，然而具体的时间值不是在母版页中产生的，而是在内容页中生成并显示在母版页上。本例在母版页中设置了一个可



读写自定义属性，然后，由内容页的代码来访问该自定义属性。下面首先列出母版页的源代码：

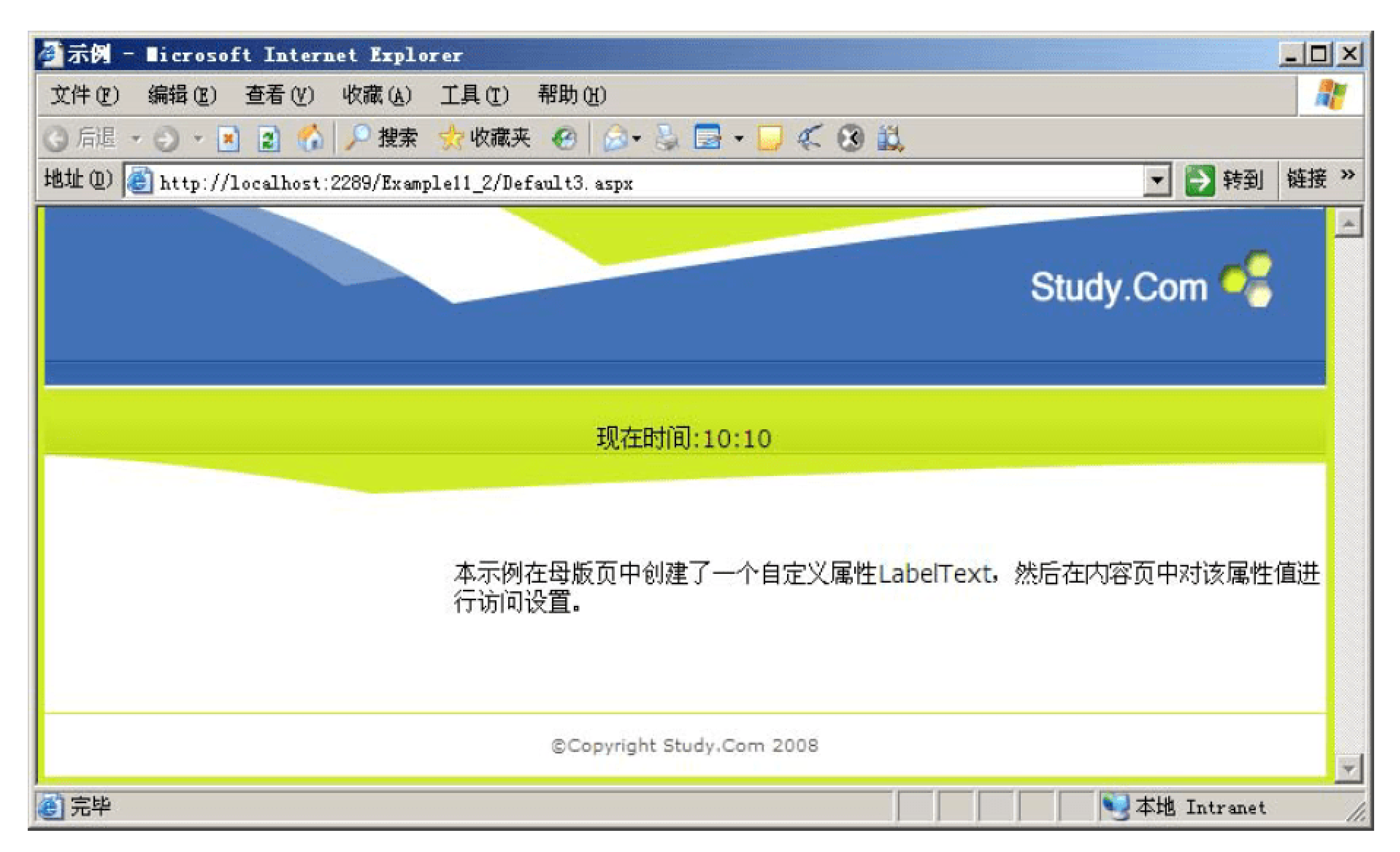


图 6-10

```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="MasterPage3.master.cs"
Inherits="MasterPage3" %>
<script runat="server">
    string _labelText = "";
    public String LabelText
    {
        get
        {
            return _labelText;
        }
        set
        {
            _labelText = value;
```

```
            }
        }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title></title>
    <link href="css/myfreetemplates.css" rel="stylesheet" type="text/css">
</head>
<body background="images/pixi_lime.gif" leftmargin="0" topmargin="0">
    <form id="form1" runat="server">
        <div align="center">
            <table width="763" height="100%" border="0" cellpadding="0"
                    cellspacing="0" bgcolor="#FFFFFF">
                <tr>
                    <td width="763" height="86" align="right" valign="top">
                        <img src="images/topic.gif"></td>
                </tr>
                <tr>
                    <td width="763" height="53" align="center" valign="bottom"
                            background="images/nav_bg.gif">
                        <%= LabelText %>
                    </td>
                </tr>
                <tr>
                    <td width="763" height="22" align="right" valign="top">
```



```
                                    <img src="images/toppic2.gif" width="763" height="22">
</td>
                        </tr>
                        <tr>
                              <td width="763" valign="top">
                                    <table width="100%" border="0" cellspacing="0"
                                            cellpadding="0">
                                          <tr>
                                                <td width="244" valign="top">
<asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
</asp:ContentPlaceHolder>
                                                </td>
                                                <td valign="top" align="left">
<asp:ContentPlaceHolder ID="ContentPlaceHolder2" runat="server">
</asp:ContentPlaceHolder>
                                                </td>
                                          </tr>
                                    </table>
                              </td>
                        </tr>
                        <tr>
                              <td width="763" height="1"
                                         background="images/pixi_lime.gif">
                                    <img src="images/pixi_lime.gif" width="1" height="1">
</td>
                        </tr>
```

```
                <tr>
                    <td width="763" height="35" align="center" class="baseline">
                        &copy;Copyright Study.Com 2008</td>
                </tr>
            </table>
        </div>
    </form>
</body>
</html>
```

上面的源代码比较简单，重点在于定义可读写属性。如粗体部分所示，与普通定义属性的方法相同，源代码中定义了一个 String 类型的属性 LabelText。LabelText 属性是一个可读写简单属性，主要用于获取和设置当前时间值，并且通过<%=LabelText %>显示在母版页中。需要注意的是，在源代码中，只是对属性进行了定义，并没有设置 LabelTextl 属性的具体值。设置 LabelText 属性值的任务是在内容页中完成的。下面列出内容页的源代码：

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage3.master" AutoEventWireup="true"
CodeFile="Default3.aspx.cs"
Inherits="Default3" Title="示例" %>

<%@ MasterType VirtualPath="~/MasterPage3.master" %>

<script runat="server">
    void Page_Load(Object sender, EventArgs e)
    {
        Master.LabelText = "现在时间:" + System.DateTime.Now.ToShortTimeString();
```



```
    }
</script>

<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
    <p> </p>
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder2" runat="server">
    <br />
    <br />
    <p>
        本示例在母版页中创建了一个自定义属性 LabelText，然后在内容页中对该属性值进行访
        问设置。</p>
    <br />
    <br />
</asp:Content>
```

上文源代码主要完成对母版页自定义属性 LabelText 的赋值。在源代码中，定义了两个代码头<% @ Page %>和<%@ MasterType %>。Page_Load 事件处理程序很简单，只是将当前时间赋予了母版页 LabelText 属性。

本例与上一示例看上去有些类似，两者的共同点是，都使用 MasterType 指令实现母版页的强类型化，从而实现对母版页公共属性的访问。两者的不同点在于，本例实现的是访问母版页的简单自定义属性，而上一例实现了访问母版页的复杂自定义属性。上一示例的本质是，通过自定义属性对母版页内部控件的封装，实现内容页对母版页控件的引用。一个是访问属性，另一个是访问控件，两者是有本质不同的。另外，在实现过程中，如果访问的是属性，或者方法，都必须将其声明为公共成员，以便进行引用。

6.1.5 动态加载母版页

上文介绍的母版页应用有一个共同特点，即内容页只绑定了一个固定的母版页，而没有实现动态加载母版页。在开发过程中，动态加载母版页的情况很多。例如，要求站点提供多个可供选择的页面模板，并允许动态加载这些模板。本节将以此应用为基础，说明动态加载母版页的具体方法。

实现动态加载母版页的核心是设置 MasterPageFile 属性值。需要强调的是，不能在 Page_Load 等事件处理程序中设置 MasterPageFile 属性，而应该在 Page_PreInit 事件处理程序中进行。因为 Page_PreInit 事件是页面生成周期中较先引发的事件。如果试图在如 Page_Load 等事件中设置 MasterPageFile 属性，将会发生页面异常。

了解以上关键内容之后，下面通过具体示例说明动态加载母版页的方法。

如图 6-11 和图 6-12 所示，两个 Web 页面共用一个内容页，而它们的母版页有差异。当用户修改下拉框选项时，Web 页面可根据选项值动态加载母版页。例如，默认情况下，内容页将绑定如图 6-11 所示的默认母版页。当选定下拉框中“动态母版页”选项时，页面将自动回传，并加载如图 6-12 所示的母版页。加载不同的母版页时，页面标题也将随之发生变化。

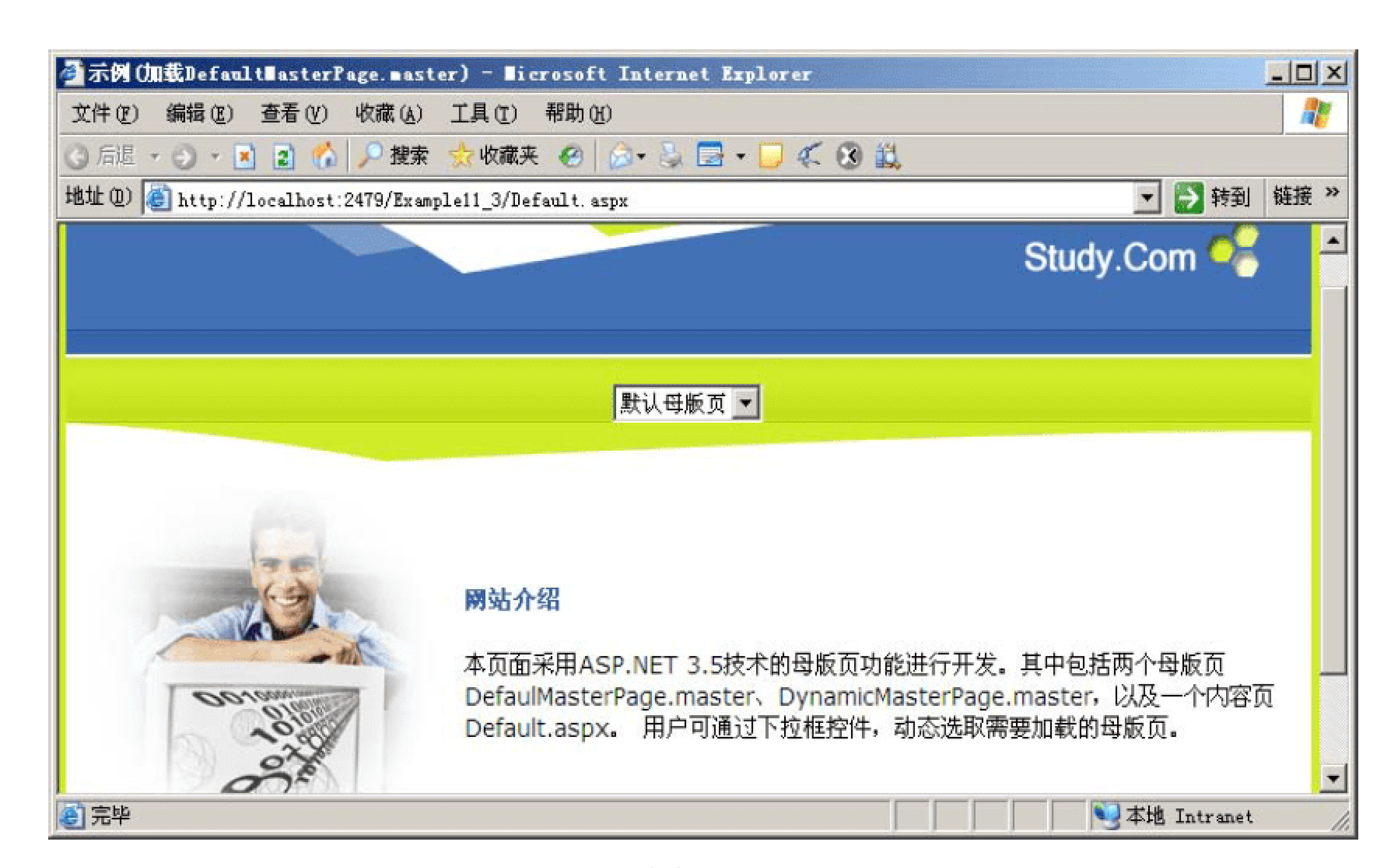
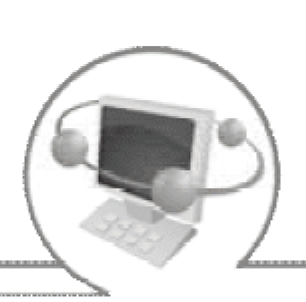


图 6-11



在实现以上应用程序之前，需要注意两个实现的要点。第一，也是最关键的，不能在 Page_Load 等事件处理程序中，实现 MasterPageFile 属性设置，而必须在 Page_PreInit 事件处理程序中加以实现。这是由母版页和内容页的初始化及加载顺序决定的。第二，由于涉及多个母版页的动态加载，因此，这些母版页具有相同的属性、方法等。基于代码重用的考虑，建议创建一个继承自 MasterPage 的自定类(这种类称为“基母版页类”)，并在其中定义母版页共用的属性和方法等特征。随后，各个母版页可继承基母版页类，由此获得共用特征。而在内容页中，如果使用 MasterType 指令强类型化母版页引用，那么可以将基母版页类作为母版页类型名称，而不是单个母版页。

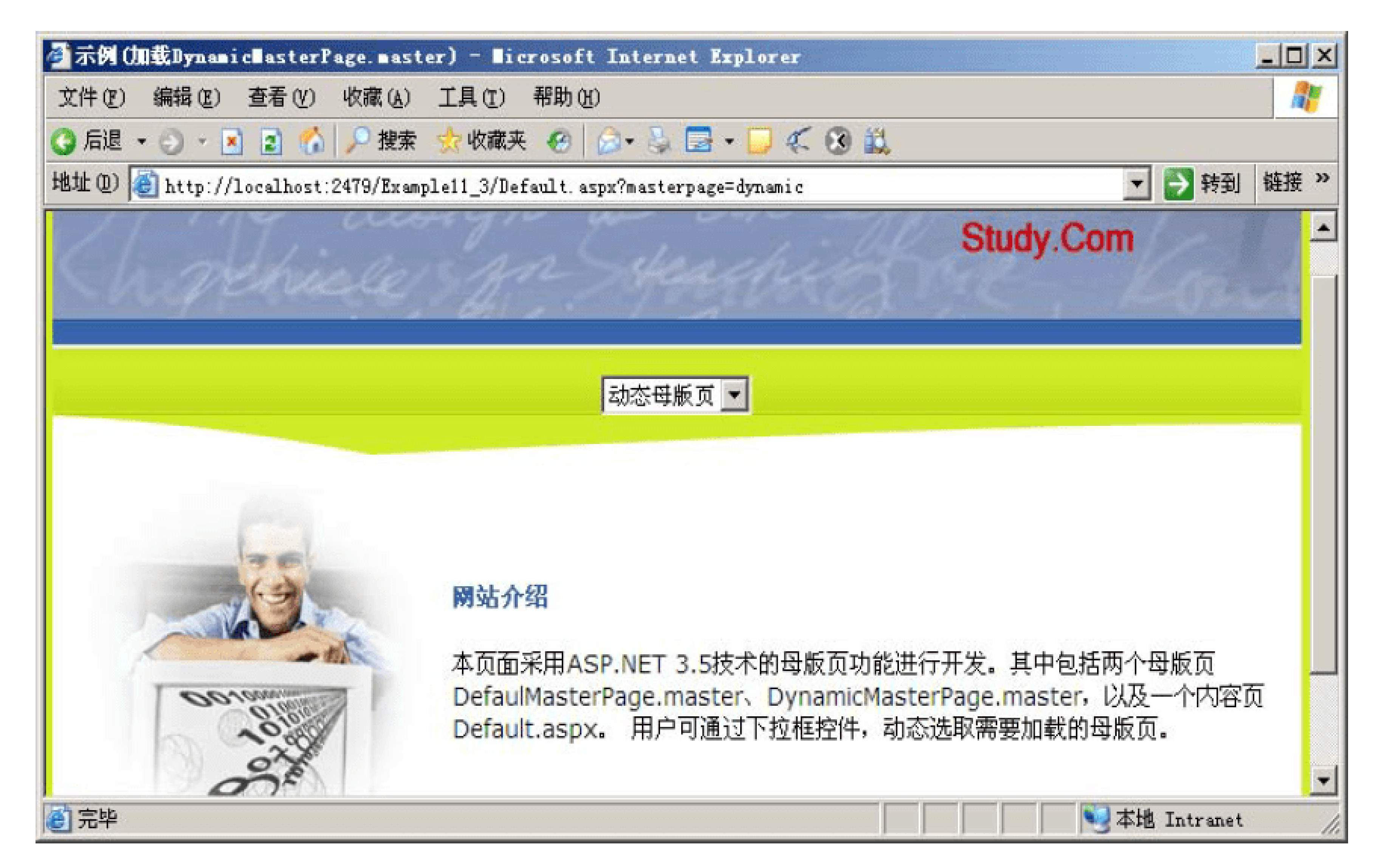


图 6-12

本示例主要包括 4 个文件，基母版页类 BaseMaster.cs、默认母版页 DefaultMasterPage.master、动态母版页 DynamicMasterPage.master 和内容页 Default.aspx。

下面列举了基母版页类文件 BaseMaster.cs 的源代码：

```
public class BaseMaster : MasterPage
{
    string _title = "";
    public virtual String TitleName
```

```
    {
        get
        {
            return _title;
        }
    }
}
```

如上代码所示，自定义类 BaseMaster 继承了 MasterPage 基类，变成了一个基母版页类。在这个基母版页类中，实现了虚拟只读属性 TitleName，并设置该属性默认值为空字串。TitleName 属性用于间接设置内容页标题文字。

下面列举了默认母版页 DefaultMasterPage.master 文件源代码：

```
<%@ Master Language="C#" Inherits="BaseMaster" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
    public override String TitleName
    {
        get
        {
            return "示例(加载 DefaultMasterPage.master)";
        }
    }
    void Page_Load(Object sender, EventArgs e)
    {
```



```
            if (!Page.IsPostBack)
            {
                string selItem = Request.QueryString["masterpage"];
                ListItem item = DropDownList1.Items.FindByValue(selItem);
                if (item != null)
                {
                    item.Selected = true;
                }
            }
        }
        void SelectedMaster(Object sender, EventArgs e)
        {
            if (DropDownList1.SelectedValue == "dynamic")
            {
                string url = Request.Path + "?masterpage=dynamic";
                Response.Redirect(url);
            }
        }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
        <title></title>
        <link href="css/myfreetemplates.css" rel="stylesheet" type="text/css">
</head>
<body background="images/pixi_lime.gif" leftmargin="0" topmargin="0">
```

```
    <form id="form1" runat="server">
        <div align="center">
            <table width="763" height="100%" border="0" cellpadding="0"
                 cellspacing="0" bgcolor="#FFFFFF">
                <tr>
                    <td width="763" height="86" align="right" valign="top">
                        <img src="images/topic.gif"></td>
                </tr>
                <tr>
                    <td width="763" height="53" align="right" valign="bottom"
                            background="images/nav_bg.gif">
                        <div style="text-align: center">
                            <asp:DropDownList ID="DropDownList1" runat="server"
                                  AutoPostBack="True" ValidationGroup="Master"
                                OnSelectedIndexChanged="SelectedMaster">
                      <asp:ListItem Value="default">默认母版页</asp:ListItem>
                      <asp:ListItem Value="dynamic">动态母版页</asp:ListItem>
                            </asp:DropDownList>
                        </div>
                    </td>
                </tr>
                <tr>
                    <td width="763" height="22" align="right" valign="top">
                        <img src="images/toppic2.gif" width="763" height="22">
</td>
                </tr>
```



```
                <tr>
                    <td width="763" valign="top">
                        <table width="100%" border="0" cellspacing="0"
                             cellpadding="0">
                            <tr>
                                <td width="244" valign="top">
<asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
</asp:ContentPlaceHolder>
                                </td>
                                <td valign="top" align="left">
<asp:ContentPlaceHolder ID="ContentPlaceHolder2" runat="server">
</asp:ContentPlaceHolder>
                                </td>
                            </tr>
                        </table>
                    </td>
                </tr>
                <tr>
                    <td width="763" height="1"
                          background="images/pixi_lime.gif">
                        <img src="images/pixi_lime.gif" width="1" height="1">
</td>
                </tr>
                <tr>
                    <td width="763" height="35" align="center" class="baseline">
                        &copy;Copyright Study.Com 2008</td>
                </tr>
```

```
                </table>
            </div>
        </form>
</body>
</html>
```

如上粗体代码所示，代码头指示该页继承自 BaseMaster 类，这样默认母版页就具有了基母版页类所定义的属性 TitleName。通过重写 TitleName 属性，为其重新设置了属性值。这样，当内容页强类型默认母版页时，在内容页中，只要使用引用的母版页对象，就能够访问重写的 TitleName 属性值，进而间接为内容页标题赋值。另外，代码中还实现了两个事件处理程序 Page_Load 和 SelectedMaster。Page_Load 事件处理程序用于设置页面加载时，下拉框控件选中项，其依据是存储于 URL 中的 QueryString 值。当下拉框控件选中项发生改变时，则引用 SelectedMaster 事件处理程序。该程序设置了当用户选中下拉框控件“动态母版页”一项时，页面的重定向 URL 地址。这为加载动态母版页时，设置下拉框控件选中项进行了参数准备。

动态母版页 DynamicMasterPage.master 文件的源代码与以上默认母版页代码相似，下面仅列出关键代码：

```
<%@ Master Language="C#" Inherits="BaseMaster" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
    public override String TitleName
    {
        get
        {
            return "示例(加载 DynamicMasterPage.master)";
```



```
            }
        }
        void Page_Load(Object sender, EventArgs e)
        {
            if (!Page.IsPostBack)
            {
                string selItem = Request.QueryString["masterpage"];
                ListItem item = DropDownList1.Items.FindByValue(selItem);
                if (item != null)
                {
                    item.Selected = true;
                }
            }
        }
        void SelectedMaster(Object sender, EventArgs e)
        {
            if (DropDownList1.SelectedValue == "default")
            {
                string url = Request.Path + "?masterpage=default";
                Response.Redirect(url);
            }
        }
</script>
```

对比默认母版页和动态母版页源代码可以发现，两者使用了同样的实现思路，从代码结构到事件处理程序基本相同。

下面列举了内容页 Default.aspx 文件的源代码：

```
<%@ Page Language="C#" %>

<%@ MasterType TypeName="BaseMaster" %>

<script runat="server">
    void Page_PreInit(Object sender, EventArgs e)
    {
        if (Request.QueryString["masterpage"] == "dynamic")
        {
            this.MasterPageFile = "DynamicMasterPage.master";
        }
        else
        {
            this.MasterPageFile = " DefaultMasterPage.master";
        }
        this.Title = Master.TitleName;
    }
</script>

<asp:content id="Content1" contentplaceholderid="ContentPlaceHolder1" runat="server">
    <img src="images/pagepic.gif" width="244" height="223" />
</asp:content>
<asp:content id="Content2" contentplaceholderid="ContentPlaceHolder2" runat="server">
    <p> </p>
    <h1>网站介绍</h1>
    <p>
        本页面采用 ASP.NET 3.5 技术的母版页功能进行开发。其中包括两个母版页
```



```
        DefaulMasterPage.master、DynamicMasterPage.master，以及一个内容页
        Default.aspx。用户可通过下拉框控件，动态选取需要加载的母版页。
    <p>
</asp:content>
```

如上粗体代码所示，代码头添加了 MasterType 指令以便对母版页实施强类型化。强类型化的目的是，使用默认母版页和动态母版页中重写的公共属性 TitleName 设置页面标题。内容页必须同时强类型化这两个母版页。按照先前示例方法，都是利用 VirtualPath 属性来指定一个需要强类型化的母版页，但是，本例中包含两个母版页，因此，VirtualPath 属性不可用。为了实现对多母版页的强类型化，可以使用 MasterType 指令中的 TypeName 属性。该属性用于设置需要强类型化的母版页类型(由此可见，VirtualPath 和 TypeName 属性不可同时使用)。由于默认母版页和动态母版页都继承自基母版页 BaseMaster，因此，可将 BaseMaster 作为 TypeName 属性值。为了动态绑定母版页，还在 Page_PreInit 事件处理程序中设置了 MasterPageFile 属性，其设置依据是 QueryString 值。

6.2　站点导航

一个 Web 站点，尤其是信息量大的大中型 Web 站点，应为用户提供站点导航。常见的导航组件包括命令菜单、树形结构、页面所处位置导航等。在 ASP.NET 1.x 时期，提供的服务器控件有限，并没有内置实现站点导航功能的控件。为此，开发人员不得不购买第三方控件，或者自行开发。然而，从这些途径获取的控件，往往存在费用昂贵、功能有限、与 Visual Studio 工具结合不紧密、执行效率低等缺点，无法满足应用程序的需要。从 ASP.NET 2.0 开始新增了站点导航控件彻底解决了这个问题。这些新增控件包括 SiteMapPath、TreeView 和 Menu 等。本节将围绕这 3 个站点导航控件，通过示例详细讲解其在 Visual Studio 2008 中的应用。

6.2.1 SiteMapPath 控件

SiteMapPath 控件用于显示一组文本或图像超链接，以便在使用最少页面空间的同时更加轻松地定位网站。

在很多大中型 Web 站点中，由于内容丰富、结构复杂，用户很容易就迷失在链接的迷宫中，经常难以准确定位当前位置以及浏览路线。SitMapPath 控件就是为有效解决这个难题而提出的。

SitMapPath 控件能够根据站点导航信息，准确定位当前页面所处整个 Web 站点的位置，同时，使用层次化表示方法，将位置信息显示为有序的静态文本或者超链接。另外，通过调整相关属性，可以自定义位置信息的外观以及其他内容，从而适应站点的总体设计风格。

SitMapPath 控件由多个节点组成，节点可分为以下 3 种类型：根节点、父节点和当前节点。表 6-1 简单说明了这些节点类型。

表 6-1　节点类型

名　称	说　明
根节点(Root)	锚定节点分层组的节点
父节点(Parent)	处于当前节点和根节点之间的是父节点，该类型节点有一个或多个子节点
当前节点(Current)	表示当前所显示页面的节点

SiteMapPath 控件的每个节点都可由 SiteMapNodeItem 对象表示。节点通常显示为静态文本或者超链接，可以为其添加自定义的外观样式或者模板。在设置模板和样式的过程中，必须注意以下两条规则：

- 如果为节点设置了自定义模板，那么模板将会自动覆盖为节点定义的任何样式。
- 为特殊类型节点设置的自定义模板和样式，将会覆盖为所有节点设置的自定义模板和样式。

与其他普通 Web 服务器控件一样，SitMapPath 的使用非常简单。可以在 Visual Studio 2008 中工具箱的“导航”选项卡下方，找到 SiteMapPath 控件。在创建基本 Web 窗体后，



使用拖放的方法，可以将 SiteMapPath 控件放置到 Web 窗体上。该控件将根据默认站点地图文件(Web.sitemap文件)中的数据，自动显示站点导航信息。这意味着必须定义好 Web.sitemap 文件的内容，然后才能使用 SiteMapPath 控件。无论是在设计时，还是运行时，SiteMapPath 控件均能够“所见即所得”地显示出站点导航信息。避免过去某些控件在设计时，显示不够直观的缺点。这一点为开发人员提供了良好的使用感受。

首先创建基本的 Web.sitemap 文件，源代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap>
  <siteMapNode title="主页" url="default.aspx">
    <siteMapNode title="计算机书籍" url="sitemapsA.aspx">
      <siteMapNode title="编程语言" url="sitemapsA1.aspx" />
      <siteMapNode title="网络应用" url="sitemapsA2.aspx" />
      <siteMapNode title="办公软件" url="sitemapsA3.aspx" />
    </siteMapNode>
    <siteMapNode title="人文类书籍" url="sitemapsB.aspx">
      <siteMapNode title="历史" url="sitemapsB4.aspx">
        <siteMapNode title="近代史" url="sitemapsB4a.aspx" />
        <siteMapNode title="现代史" url="sitemapsB4b.aspx" />
      </siteMapNode>
      <siteMapNode title="经济" url="sitemapsB5.aspx" />
      <siteMapNode title="教育" url="sitemapsB6.aspx" />
    </siteMapNode>
    <siteMapNode title="自然科学书籍" url="sitemapsC.aspx" />
  </siteMapNode>
</siteMap>
```

在源代码中，定义了一组简单的站点导航信息，说明了 Web 站点的逻辑结构。该 Web

站点的根节点中“主页”的 URL 地址是 default.aspx。另外，还包含其他多个节点，例如，“人文类书籍”、“历史”、“经济”等。每个节点都与相关 Web 页面相关联。

本例创建的是与节点“近代史”相关联的 Web 页面 sitemapsB4a.aspx。重点是在该页面中实现 SiteMapPath 控件，并且设置多个 SiteMapPath 控件属性，说明这些属性的使用方法。本例的效果如图 6-13 所示。



图 6-13

如图 6-13 所示，表格中共包括 6 个 SiteMapPath 控件的具体实例。sitemapsB4a.aspx 源代码如下所示：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="sitemapsB4a.aspx.cs"
Inherits="sitemapsB4a" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>示例</title>
<link id="InstanceStyle" href="StyleSheet.css"
type="text/css" rel="stylesheet" />
</head>
<body>
```



```
<form id="form1" runat="server">
    <div>
        <fieldset style="width: 460px">
            <legend class="mainTitle">SiteMapPath 控件基本应用</legend>
            <table bordercolor="black" cellspacing="0"
                    bordercolordark="white" cellpadding="5"
                border="1">
                <tr>
                    <td style="text-align: center; font-weight: bold;
                          background-color: lavender;" width="220">
                        属性设置
                    </td>
                    <td style="text-align: center; font-weight: bold;
                            background-color: lavender;" width="300">
                        效果 </td>
                </tr>
                <tr>
                    <td>
                        未设置属性
                    </td>
                    <td>
                        <asp:SiteMapPath ID="SiteMapPath1" runat="server">
                        </asp:SiteMapPath>
                    </td>
                </tr>
                <tr>
```

```
            <td>
                Colorful 格式</td>
            <td>
                <asp:SiteMapPath ID="SiteMapPath2" runat="server"
                        PathSeparator=" : " Font-Names="Verdana"
                    Font-Size="0.8em">
                    <PathSeparatorStyle Font-Bold="True"
                          ForeColor="#990000"></PathSeparatorStyle>
                    <CurrentNodeStyle ForeColor="#333333" />
                    <NodeStyle Font-Bold="True"
                        ForeColor="#990000"></NodeStyle>
                    <RootNodeStyle Font-Bold="True"
                        ForeColor="#FF8000"></RootNodeStyle>
                </asp:SiteMapPath>
            </td>
        </tr>
        <tr>
            <td>
                PathSeparator=" | "
            </td>
            <td>
                <asp:SiteMapPath ID="SiteMapPath3" runat="server"
                      PathSeparator=" | ">
                </asp:SiteMapPath>
            </td>
        </tr>
```



```
<tr>
    <td>
        PathDirection="CurrentToRoot"
    </td>
    <td>
        <asp:SiteMapPath ID="SiteMapPath4" runat="server"
            PathDirection="CurrentToRoot">
        </asp:SiteMapPath>
    </td>
</tr>
<tr>
    <td>
        ParentLevelsDisplayed="2"
    </td>
    <td>
        <asp:SiteMapPath ID="SiteMapPath5" runat="server"
             ParentLevelsDisplayed="2">
        </asp:SiteMapPath>
    </td>
</tr>
<tr>
    <td>
        RenderCurrentNodeAsLink="True"
    </td>
    <td>
        <asp:SiteMapPath ID="SiteMapPath6" runat="server"
```

```
                        RenderCurrentNodeAsLink="True">
                    </asp:SiteMapPath>
                </td>
            </tr>
        </table>
    </fieldset>
  </div>
 </form>
</body>
</html>
```

SiteMapPath 控件的定义是非常简单的，只要在.aspx 文件中，输入以下代码，就能够轻松定义 SiteMapPath 控件。

```
<asp:SiteMapPath ID="SiteMapPath1" Runat="server"></asp:SiteMapPath>
```

多数与数据相关的服务器控件，都需要数据源控件的支持。由于 SiteMapPath 控件默认支持 Web.sitemap 文件作为站点导航信息的数据存储，因此，只需在应用程序中定义 Web.sitemap 文件内容，并且将 SiteMapPath 控件置于 Web 页面中，SiteMapPath 控件就能够自动显示导航信息。

如上源代码所示，可以通过定义一些属性来设置 SiteMapPath 控件的外观样式、行为等特征。本例中列举了 5 个属性设置的实例，实际上，这些属性的设置并不是使用属性窗口一一定义的结果，而是另有他法。将 SiteMapPath 控件拖放到 Visual Studio 2008 的 Web 窗体之上后，将会自动出现如图 6-14 所示的控件和任务列表。



图 6-14



在任务列表中，包含了两个常用任务：自动套用格式和编辑模板。前者用于定义控件格式，后者用于编辑模板。单击“自动套用格式”选项后，将出现如图 6-15 所示的对话框。



图 6-15

Visual Studio 2008 定义了 5 种 SiteMapPath 控件常用格式：移除格式设置、彩色型、简明型、专业型和传统型。选中左栏选项，右栏将同时出现预览效果。开发人员可以在整个 Web 应用程序中，使用统一的 SiteMapPath 控件外观。

6.2.2　TreeView 控件

要在 Web 站点中实现与 TreeView 类似的功能，必须在客户端编写大量的 DHTML，以使其正常工作。这是非常痛苦的一件事，因为，必须要考虑各种 Web 浏览器及其各个版本的所有特征，且测试将比较繁琐。在 ASP.NET 中，所有这些复杂性都不存在了，它们被封装在新增的 TreeView 控件中。本小节将对 TreeView 控件的功能特征和对象结构等进行介绍。

TreeView 控件在客户端浏览器中会显示一个树形结构，此树与 Windows 资源管理器中的树非常类似。该控件主要用来显示分级数据，例如，目录结构数据、多层表结构数据等。开发人员可以通过以下列出的功能特征，了解 TreeView 控件所具有的功能。

- 支持数据绑定。即允许通过数据绑定方式，使得控件节点与 XML、表格、关系型数据等结构化数据建立紧密联系。
- 支持站点导航功能。即通过集成 SiteMapDataSource 控件，实现站点导航功能。
- 节点文字可显示为普通文本或者超链接文本。
- 支持动态构建功能。通过编程访问 TreeView 对象模型，完成动态创建树形结构、构建节点和设置属性等任务。
- 在客户端浏览器支持的情况下，支持由客户端构建节点，以此减少到服务器端的回送。
- 具有在节点显示 CheckBox 控件的功能。
- 可自定义树形和节点的样式、主题等外观特征。
- 可根据不同类型设备和浏览器，自适应地完成控件呈现。

TreeView 由任意多个 TreeNode 对象组成，每个 TreeNode 还可以包括任意多个子 TreeNode 对象。包含 TreeNode 及其子节点的层次结构构成了 TreeView 控件所呈现的树结构。表 6-2 显示的是节点类型列表。

表 6-2　节点类型

名　称	说　明
根节点(Root)	处于树形最顶层，其与一个或多个子节点相连
父节点(Parent)	该类型节点有一个父节点，并且与一个或者多个子节点相连
叶节点(Leaf)	该类型节点处于树形最下层，其无子节点

图 6-16 所示显示了 TreeView 控件的节点类型。

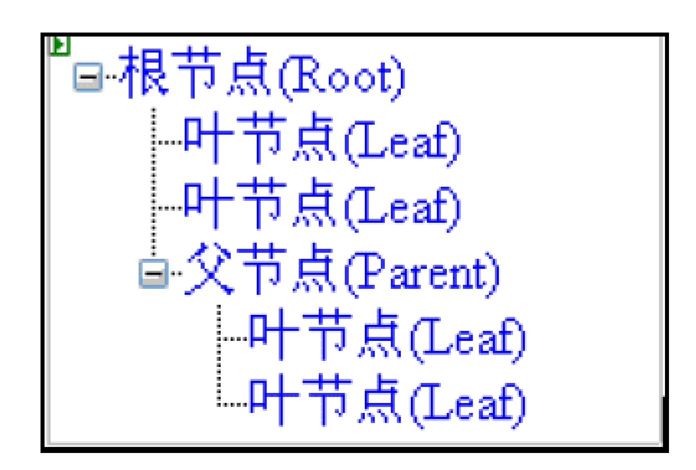


图 6-16



除图中所示的 3 种节点类型外，还有一种常见的节点——子节点。子节点是指该节点处于另一节点的下层。图 6-16 中，所有叶节点和父节点均可称为子节点。另外，图中显示的树形结构只有一个根节点，这是 TreeView 控件最为典型的结构。实际上，TreeView 允许创建具有多个根节点的树形结构，这种多根节点的树形还是比较常见的。

TreeView 控件的基本功能可以总结为：将有序的层次化数据显示为树形结构。这一功能可以为用户提供极大便利。下面我们就将以 TreeView 控件的基本应用为中心，举例说明为 TreeView 控件添加节点和设置属性的方法。示例效果如图 6-17 所示。

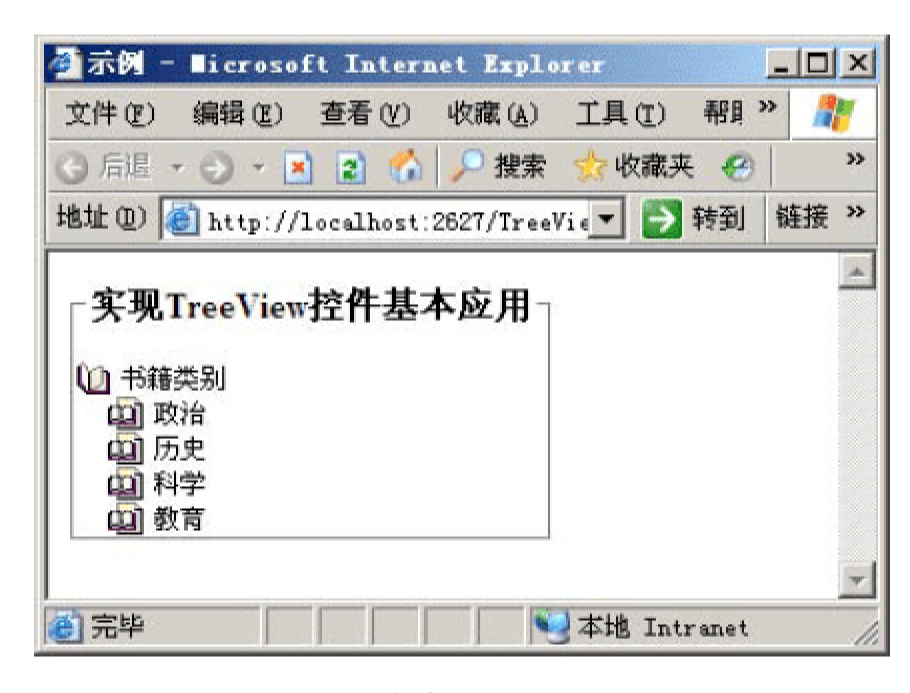


图 6-17

图中包含一个基本的 TreeView 控件。在树形结构中，包括 1 个根节点和 4 个子节点。控件被设置为类似 Windows 帮助文件的外观样式。

实现本例主要涉及两大步骤：一是为 TreeView 控件添加自定义节点，二是设置控件外观属性。与其他服务器控件应用相同，在创建基本 Web 窗体之后，可以使用拖放的方法，将 TreeView 控件拖放到 Web 窗体的适当位置。此时，在 Web 文件的设计窗口，将会出现如图 6-18 所示的控件和任务列表。



图 6-18

在任务列表中，显示的是设置 TreeView 控件的常用任务，通常情况下，先定义节点

后设置外观样式。首先要对 TreeView 控件定义节点。单击编辑节点，将会弹出如图 6-19 所示的对话框。该对话框用于定义 TreeView 控件的节点和相关属性。当前选中节点是名为“书籍类别”的根节点，并且分别对其 Text、Value 和 NavigateUrl 属性进行了设置。其他节点也可进行类似设置。

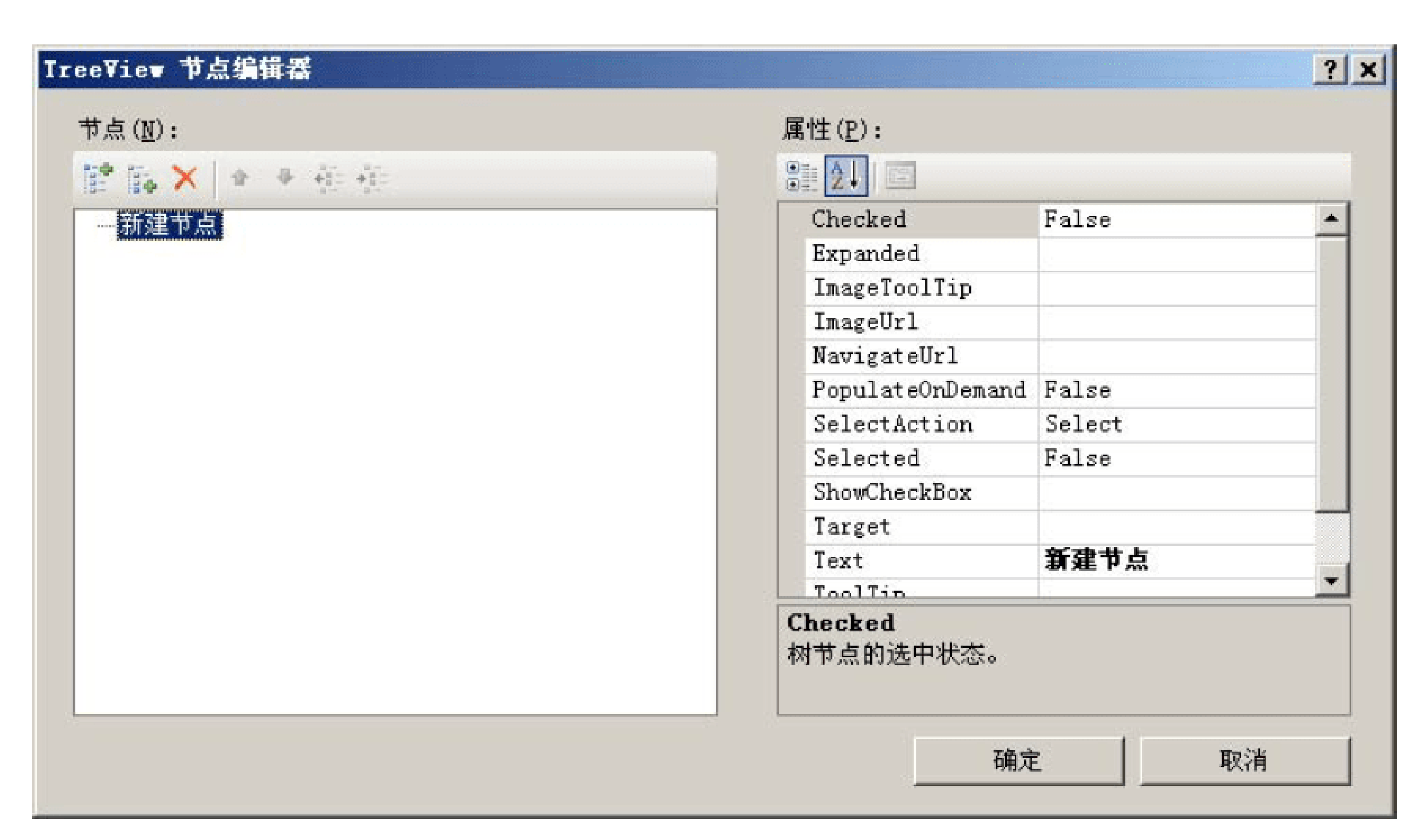


图 6-19

以上设置过程非常简单。有关源代码如下所示：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>


<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>示例</title>
<link id="InstanceStyle" href="StyleSheet.css" type="text/css"
rel="stylesheet" />
</head>
<body>
    <form id="form1" runat="server">
        <div>
```



```
        <fieldset style="width: 140px">
            <legend class="mainTitle">实现 TreeView 控件基本应用</legend>
            <br />
            <asp:TreeView ID="TreeView2" runat="server"
ImageSet="WindowsHelp" NodeIndent="15">
                <SelectedNodeStyle BackColor="#B5B5B5"></SelectedNodeStyle>
                <Nodes>
                    <asp:TreeNode Value="书籍类别"
                            NavigateUrl="Default.aspx" Text="书籍类别">
                        <asp:TreeNode Value="政治" NavigateUrl="Page1.aspx"
                            Text="政治"></asp:TreeNode>
                        <asp:TreeNode Value="历史" NavigateUrl="Page2.aspx"
                            Text="历史"></asp:TreeNode>
                        <asp:TreeNode Value="科学" NavigateUrl="Page3.aspx"
                            Text="科学"></asp:TreeNode>
                        <asp:TreeNode Value="教育" NavigateUrl="Page4.aspx"
                            Text="教育"></asp:TreeNode>
                    </asp:TreeNode>
                </Nodes>
                <NodeStyle VerticalPadding="1" Font-Names="Tahoma"
                            Font-Size="8pt" HorizontalPadding="5"
                    ForeColor="Black"></NodeStyle>
                <HoverNodeStyle Font-Underline="True"
                            ForeColor="#6666AA"></HoverNodeStyle>
            </asp:TreeView>
        </fieldset>
```

```
            </div>
        </form>
</body>
</html>
```

虽然 TreeView 控件的节点定义和外观属性设置比较简单，甚至在不需要编写一行代码的情况下完成，但是，作为开发人员，仍然需要熟悉 TreeView 控件的源代码。上文所示粗体文字是 TreeView 控件的相关代码，其中包括外观样式属性、节点设置等多方面内容。

由代码可知，TreeView 控件的节点定义完全包含在标签<Nodes>中，<Nodes>中包含的所有内容组成了 TreeView 控件的节点集合。无论何种类型的节点，均采用<asp:TreeNode>标签进行定义。对于父子关系的节点，可采用标签嵌套的方法，进行代码定义。

6.2.3 Menu 控件

在 Web 应用程序中，虽然构建菜单的难度有些大，但是还存在一些方法。典型的如使用 JavaScript、CSS、DHTML 等技术构建导航菜单。由于 Web 中的菜单包含很多客户端交互功能和精确的外观样式，因此，开发人员就将有关功能和样式封装入 JavaScript、CSS 等文件中。使用这种方法虽然能够实现 Web 菜单，但是存在如构建难度大、灵活性差等缺点。ASP.NET 提供了 Menu 控件来解决这一难题。

根据菜单项显示方式，Menu 控件可分为两种类型：静态菜单和动态菜单。静态菜单是指菜单选项始终不变。默认情况下，位于 Menu 控件根部的根菜单项就是一种典型的静态菜单，通过属性设置，可以控制静态菜单的显示级数。动态菜单是指当用户鼠标位于菜单项上方时，在其水平或者垂直方向将会自动显示子菜单项。子菜单项在显示短暂时间后，将自动消失。通过设置属性值，可以调整子菜单的延迟时间。

Menu 控件由多个菜单项组成。这些菜单项分为 3 种类型：位于控件最顶层的是根菜单项，Menu 控件可包含多个根菜单项。如果菜单项包含其他菜单项，则称其为父菜单项，而其下层的称为子菜单项。

Menu 控件的基本功能是实现站点导航，在此基础上，该控件还支持以下高级功能：



- 支持数据绑定。即允许通过数据绑定方式，使得菜单选项与 XML、表格等结构化数据建立紧密联系。
- 通过集成 SiteMapDataSource 控件，实现站点导航功能。
- 支持动态构建功能。通过编程访问 Menu 对象模型，完成动态创建菜单、构建菜单选项和设置属性等任务。
- 可使用主题、样式属性、模板等，实现自定义控件外观。
- 可根据不同类型的设备和浏览器，自适应地完成控件呈现。

下面我们将围绕 Menu 控件的基本应用，举例说明为 Menu 控件手动创建菜单和应用样式的方法。示例效果如图 6-20 所示。



图 6-20

图中的 Menu 控件包含多个菜单项以及外观样式。当鼠标位于根菜单项“个人文件”时，将自动显示子菜单项“日常工作”和“我的文档”。当鼠标继续移动至菜单项“我的文档”上时，将自动显示子菜单项“文件存储”和“电子邮件”。

实现本例主要涉及两大步骤：一是为 Menu 控件手动添加自定义菜单项，二是设置控件外观属性。与其他服务器控件应用相同，在创建基本 Web 窗体之后，可以使用拖放的方法，将 Menu 控件拖放到 Web 窗体的适当位置。此时，在 Web 文件的设计窗口，将会出现如图 6-21 所示的 Menu 控件和任务列表。



图 6-21

在任务列表中，单击“编辑菜单项”将弹出如图 6-22 所示的对话框。

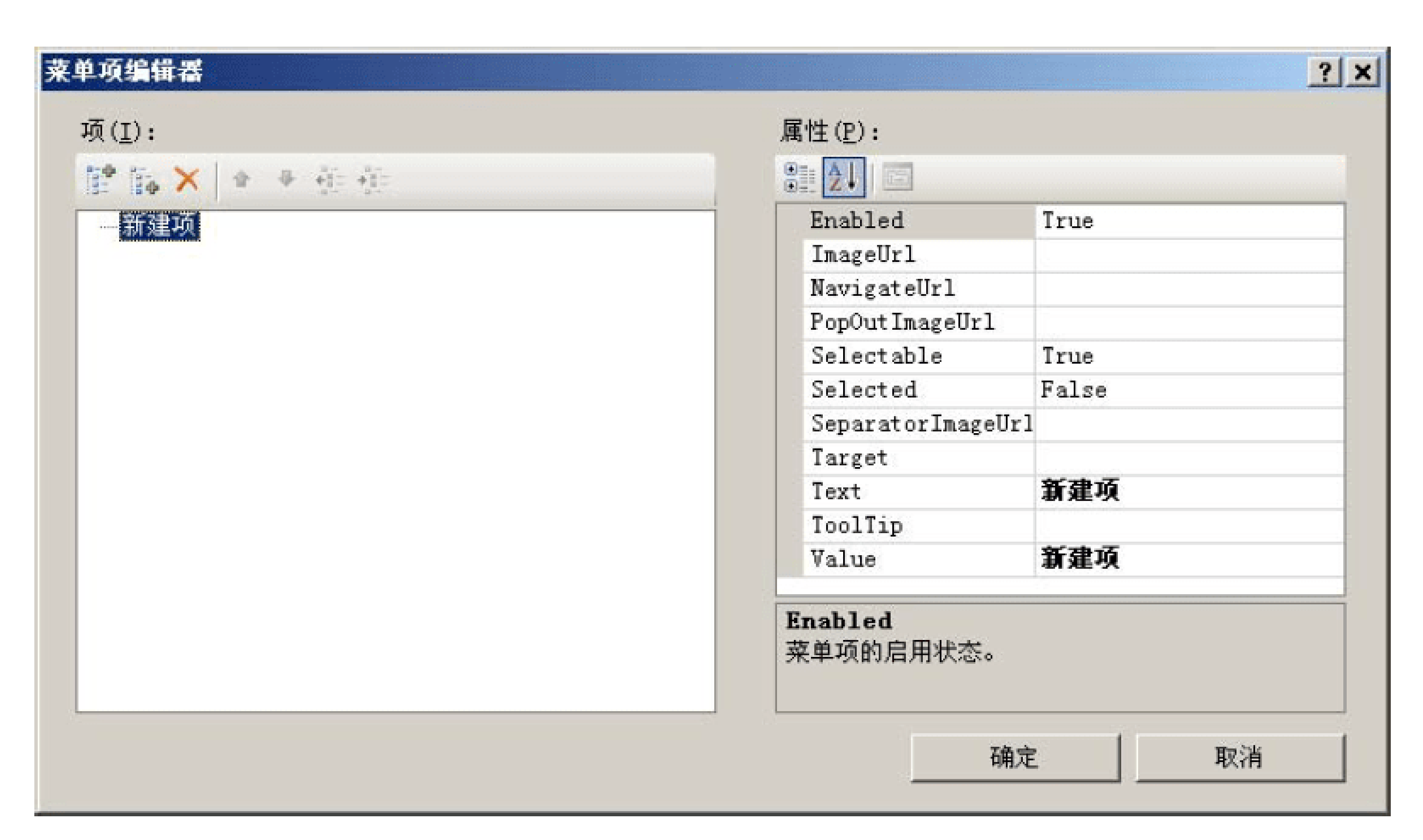


图 6-22

该对话框用于自定义 Menu 控件菜单项内容及其相关属性。对话框左列包括操作菜单项的命令按钮和控件预览窗口。使用这些命令按钮，可以实现对 Menu 控件菜单项的添加、删除、调整位置等操作。对话框右侧显示的是当前选中菜单项的属性列表。根据应用需要，可逐个设置菜单项属性。Menu 控件中设置了两个根菜单项以及多个子菜单项，并且对这些菜单项进行了属性设置。至此，就完成了对 Menu 控件手动添加菜单项的任务。

以上设置过程非常简单。有关源代码如下所示：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>示例</title>
<link id="InstanceStyle" href="StyleSheet.css" type="text/css"
rel="stylesheet" />
</head>
<body>
    <form id="form1" runat="server">
```



```
<div>
    <fieldset style="width: 240px">
        <legend class="mainTitle">实现 Menu 控件基本应用</legend><br />
        <asp:Menu ID="Menu1" runat="server" BackColor="#B5C7DE"
                DynamicHorizontalOffset="2" Font-Size="10pt"
                ForeColor="#284E98" StaticSubMenuIndent="10px">
            <DynamicHoverStyle Font-Bold="True" BackColor="#284E98"
                   ForeColor="White"></DynamicHoverStyle>
            <Items>
                <asp:MenuItem Value="系统配置" Text="系统配置" />
                <asp:MenuItem Value="个人文件" Text="个人文件">
                    <asp:MenuItem Value="日常工作" Text="日常工作" />
                    <asp:MenuItem Value="我的文档" Text="我的文档">
                        <asp:MenuItem Value="文件存储" Text="文件存储"/>
                        <asp:MenuItem Value="电子邮件" Text="电子邮件"/>
                    </asp:MenuItem>
                </asp:MenuItem>
            </Items>
            <StaticHoverStyle Font-Bold="True" BackColor="#284E98"
                      ForeColor="White"></StaticHoverStyle>
            <StaticMenuItemStyle HorizontalPadding="5px"
                      VerticalPadding="2px" />
            <DynamicMenuStyle BackColor="#B5C7DE" />
            <StaticSelectedStyle BackColor="#507CD1" />
            <DynamicSelectedStyle BackColor="#507CD1" />
            <DynamicMenuItemStyle HorizontalPadding="5px"
```

```
VerticalPadding="2px" />
                    </asp:Menu>
                    <br /><br />
                </fieldset>
            </div>
        </form>
</body>
</html>
```

粗体部分是 Menu 控件的定义源代码，主要包含两个部分：一是菜单项定义代码，二是外观样式定义代码。由代码可知，Menu 控件的所有菜单项定义代码，均置于标签<Items>中，如果要定义一个菜单项，只需编写<asp:MenuItem>标签，并且设置相关属性。通常情况下，至少要为菜单项定义 Text 和 Value 属性。其中 Text 属性用于设置菜单项的显示文字，Value 属性用于设置菜单项表示的值。另外，菜单项之间的关系，是通过<asp:MenuItem>标签之间的嵌套关系来实现的。

【小结】

- 创建母版页和内容页。
- 访问母版页。
- 动态加载母版页。
- 使用 SiteMapPath 控件、TreeView 控件、Menu 控件。



【自测题】

1. 母版页的后缀名是(　　)。

   A. .aspx　　　　B. .ascx

   C. .master　　　D. .skin

2. “客户端浏览器可以向服务器发出请求，直接访问母版面”，这种说法(　　)。

   A. 对　　　　B. 错

3. 要访问母版页的控件和属性可通过(　　)。

   A. FindControl　　B. MasterType 指令　　C. 直接获取

4. SiteMapPath 控件的节点类型有(　　)。

   A. Root　　　　B. Parent

   C. Current　　D. SubNode

5. TreeView 控件的节点类型有(　　)。

   A. Root　　　　B. Parent

   C. SubNode　　D. Leaf

【上机部分】

上机目标

- 使用母版页
- 访问母版页
- 站点导航控件的使用



上机练习

◆ 第一阶段 ◆

练习 1：创建母版页。

【问题描述】

母版页的运行技术虽然复杂，但是对于母版页和内容页本身的技术使用，实际上是很容易的。本阶段将练习如何通过 Visual Studio 2008 创建母版页。

【问题分析】

使用 Visual Studio 2008 创建母版页。

【参考步骤】

(1) 启动 Visual Studio 2008，新建一个网站。

(2) 右键单击“网站”项目，并从快捷菜单中选择“添加新项”命令，弹出“添加新项”对话框，单击“母版页文件”类型，将该母版页命名为 MasterPage.master，单击“添加”按钮。

(3) 在源视图编辑状态下，可以看到文件第一行标有@Master 指令的 HTML 代码，如下所示:

```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="MasterPage.master.cs"
Inherits="MasterPage" %>
```

母版页文件建立完毕之后，页面上会自动生成一个 contentplaceholder 控件，如下代码所示:

```
<asp:contentplaceholder id="ContentPlaceHolder1" runat="server"> </asp:contentplaceholder>
```

(4) 在源视图编辑状态下，添加用于统一布局的 HTML 代码以及内容占位符控件。将下面的代码写入到 body 之间，写入后完整的母版页 HTML 代码如下所示:



```
<body>
    <form id="form1" runat="server">
    <div>
        <table border="1">
            <tr>
                <td colspan=2>
                    <strong>
<span style="font-size:10pt;font-family:Tahoma">
网站页眉部分，这部分对于所有内容页面都是不可编辑的
</span>
</strong>
                </td>
            </tr>
            <tr>
            <td valign="top" style="width:189px">
                <span style="font-size:10pt;font-family:Tahoma">
导航条部分<br />
                1.<a href="default.aspx">网站首页</a><br />
                2.<a href="Login.aspx">用户登录</a><br />
                3.<a href="OtherLink.aspx">相关链接</a>
                </span>
            </td>
            <td style="width:805px">
<asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
     <span style="font-size:10pt;font-family:Tahoma">
这部分是内容页面展示部分，它显示了每个导航页面单击之后的内容页面内容
```

```
</span>
</asp:ContentPlaceHolder>
                </td>
                </tr>
            </table>
            <br />
            <br />
             <br />
    </div>
    </form>
</body>
```

(5) 代码添加完毕之后，将文件切换到设计视图，可以看到该母版页套用布局之后的设计页面，如图 6-23 所示。

图 6-23

练习 2：创建内容页。

【问题描述】

创建内容页。

【问题分析】

练习 1 中创建了母版页，但母版页并不能直接显示，需要创建内容页。内容页的创建同一般的 aspx 页面的创建类似，只是在页面创建之后，需要设定 Content 控件的属性，使



其与母版页绑定。

【参考步骤】

(1) 右键单击“网站”项目，并从快捷菜单中选择“添加新项”命令，弹出“添加新项”对话框，选择“Web 窗体”文件类型并勾选“选择母版页”复选框，同时给该页面命名为 Default.aspx，单击“添加”按钮并选择“MasterPage.master”为母版页后完成页面的新建。

(2) 重复第 1 步，再分别建立 OtherLink.aspx 文件和 Login.aspx 文件。

(3) 双击 Default.aspx 文件，打开该页面的源编辑视图，用下面的代码覆盖原有 HTML 代码:

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.master" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
<span style="font-size:16pt;font-family:Tahoma">
<strong>网站首页<br />欢迎进入我的站点！！！</strong>
</span>
</asp:Content>
```

代码中的 MasterPageFile="~/MasterPage.master"建立了 Default.aspx 内容页与母版页之间的使用关系，而 ContentPlaceHolderID="ContentPlaceHolder1"指明了该内容页会替换母版页中 ContentPlaceHolder1 占位符控件所占据的母版页位置。

(4) 双击 OtherLink.aspx 文件，打开该页面的源编辑视图，用下面的代码覆盖原有 HTML 代码。

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.master" AutoEventWireup="true"
CodeFile="OtherLink.aspx.cs" Inherits="OtherLink" Title="Untitled Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
    <span style="font-size:16pt;font-family:Tahoma">
        <strong>相关链接</strong>
```

```
    </span>
</asp:Content>
```

(5) 双击 Login.aspx 文件，打开该页面的源编辑视图，用下面的代码覆盖原有 HTML 代码。

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.master" AutoEventWireup="true"
CodeFile="Login.aspx.cs" Inherits="Login" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
    <span style="font-size:16pt;font-family:Tahoma">
        <strong>用户登录<br />
        <asp:Label ID="Label1" runat="server" Text="用户名"></asp:Label>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <br />
        <asp:Label ID="Label2" runat="server" Text="密码"></asp:Label>
        <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
        <asp:Button ID="Button1" runat="server" Text="登录" />
        </strong>
</span>
</asp:Content>
```

(6) 右键单击 Default.aspx 文件，选择“在浏览器中查看”菜单，运行效果如图 6-24～图 6-26 所示。

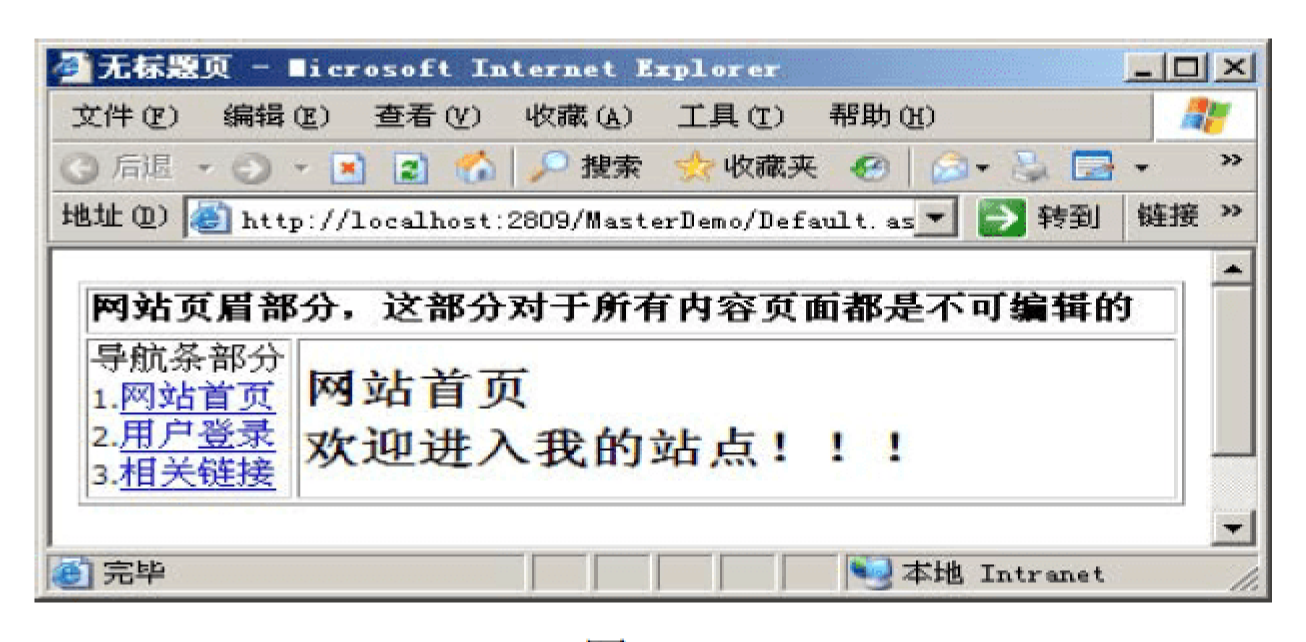


图 6-24



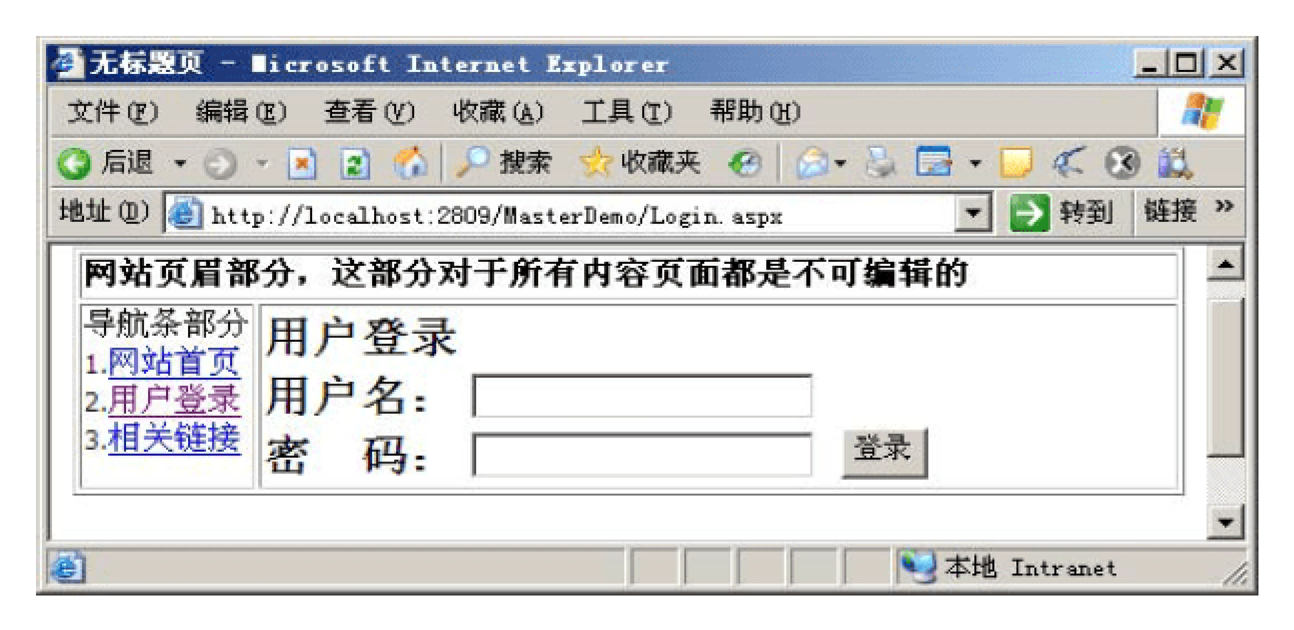


图 6-25

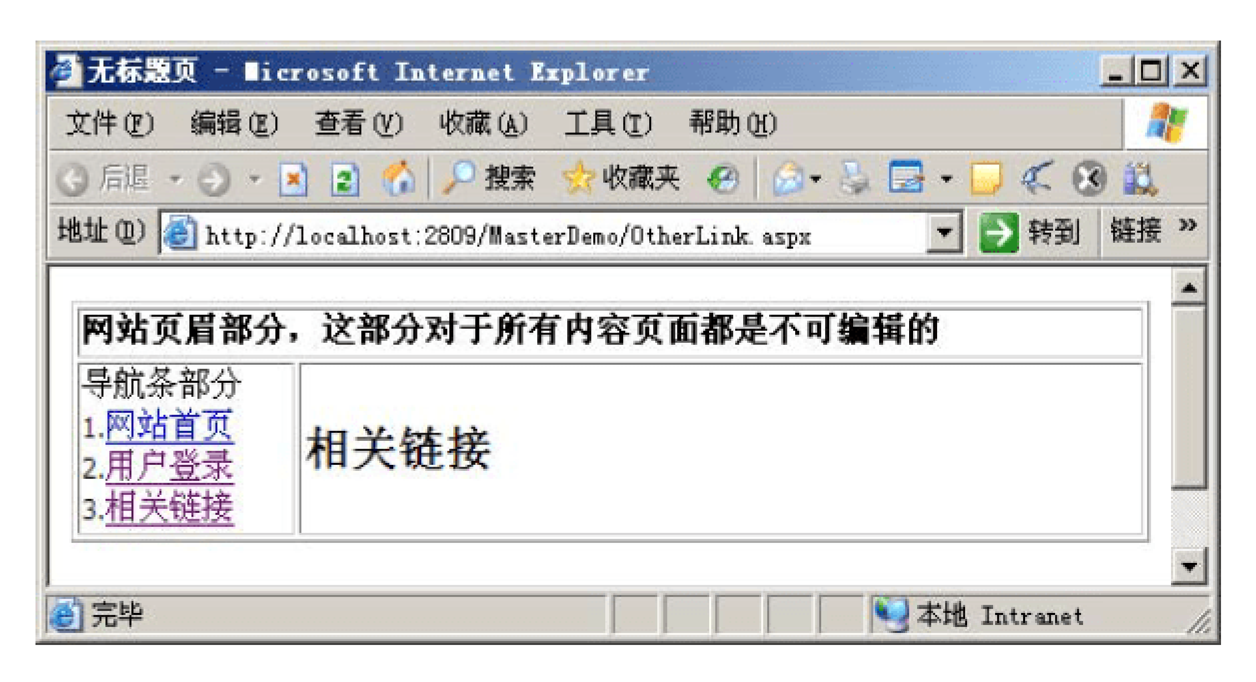


图 6-26

◆ 第二阶段 ◆

练习 1：TreeView 控件绑定 Web.sitemap 文件。

【问题描述】

本示例使用 Web.sitmap 文件与 TreeView 控件相结合，显示站点导航信息。示例效果如图 6-27 所示。

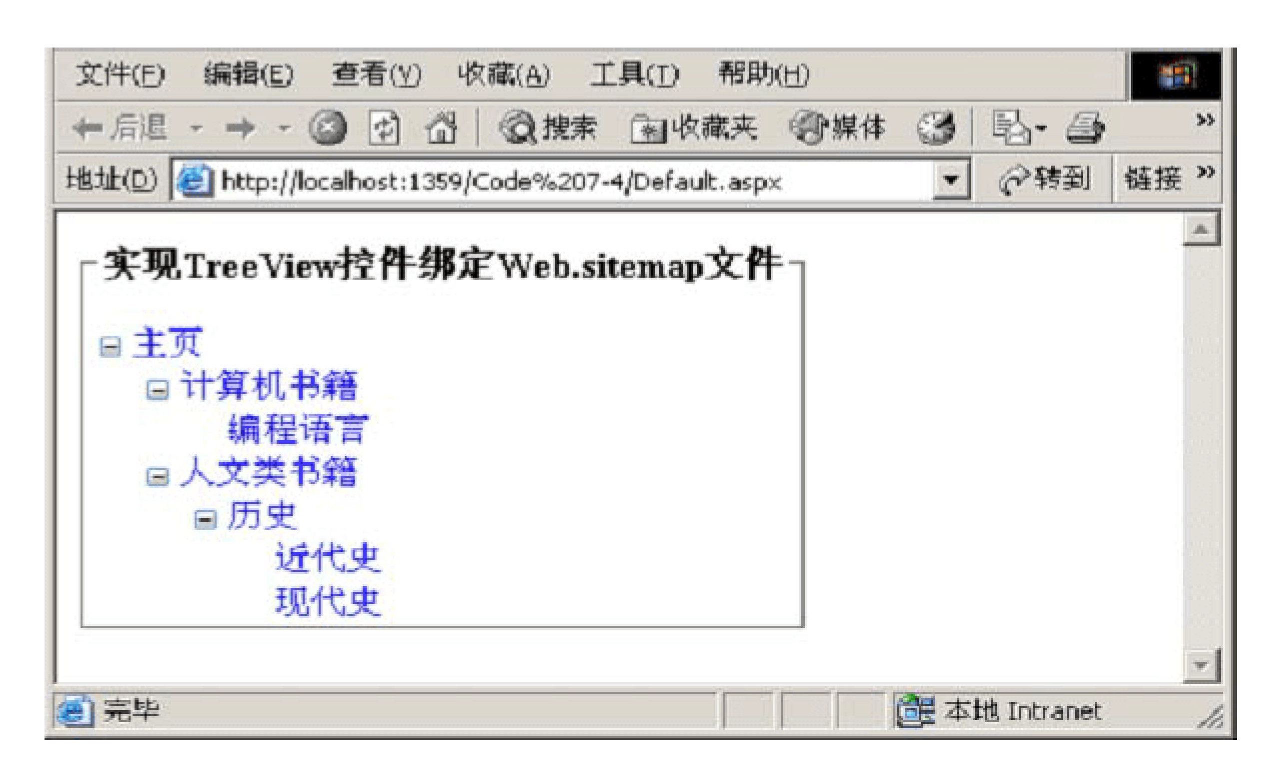


图 6-27

【问题分析】

Web.sitemap 文件是站点导航信息的数据存储，其数据采用普通 XML 格式，将站点逻辑结构层次化地列出。Web.sitemap 与 TreeView 控件集成的实质是以 Web.sitemap 文件为数据基础，以 TreeView 控件的树形结构为表现形式，将站点的逻辑结构表现出来。

示例图中显示的是一个由 TreeView 控件构建的树形结构，每个节点都是一个超链接。单击节点的超链接，可以导航到相关 Web 页面。实现以上示例，主要包括两个关键步骤：一是创建树形结构的数据基础——Web.sitemap 文件；二是设置 TreeView 控件属性。

【参考步骤】

(1) 启动 Visual Studio 2008，新建一个网站。

(2) 在“解决方案资源管理器”上的站点上单击鼠标右键，选择添加新项。出现图 6-28 所示的对话框。

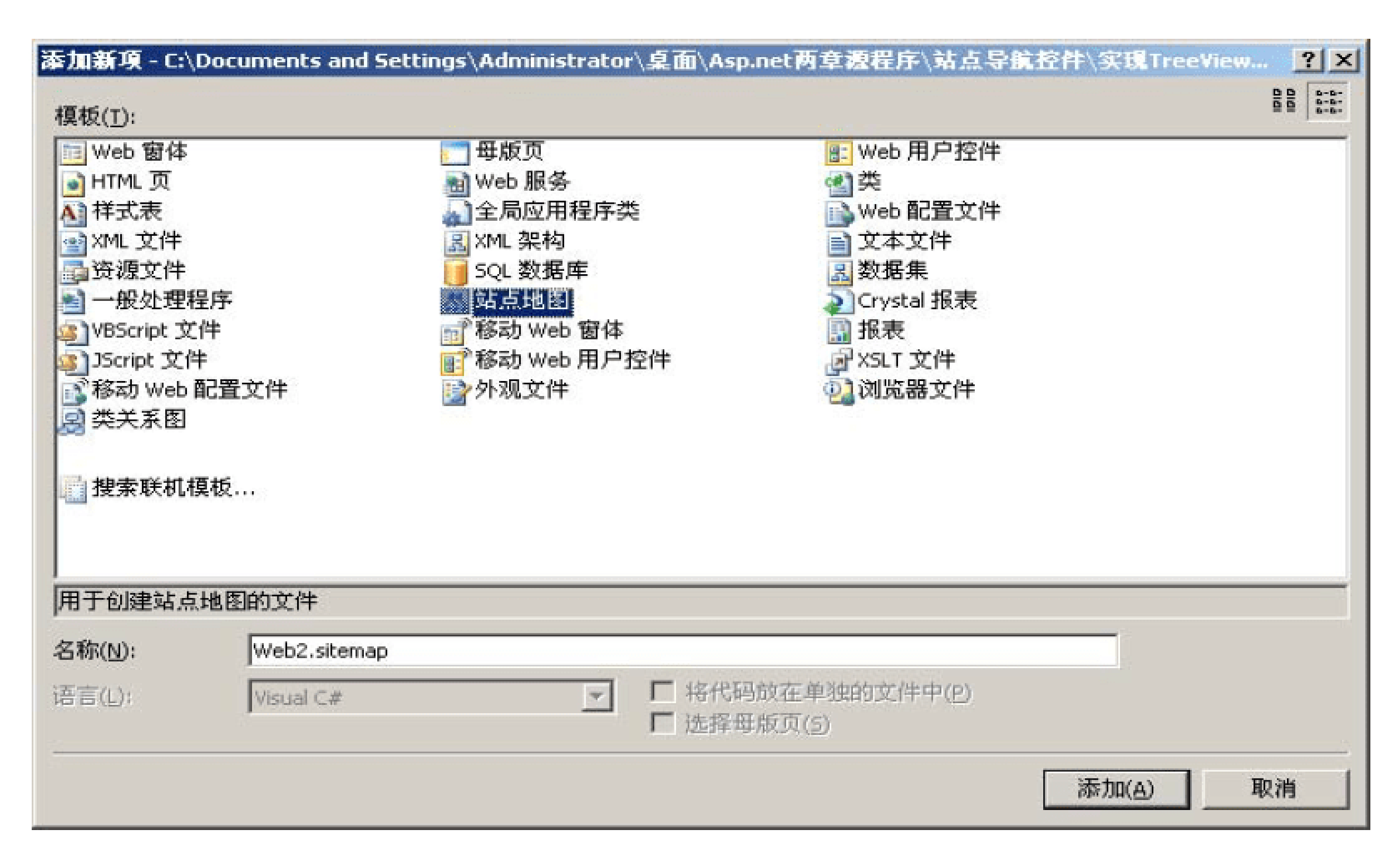


图 6-28

(3) 在“添加新项”对话框中，选择“站点地图”，新建一站点文件 Web.sitemap。

(4) Web.sitemap 文件源代码如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap>
    <siteMapNode title="主页" url="default.aspx">
```

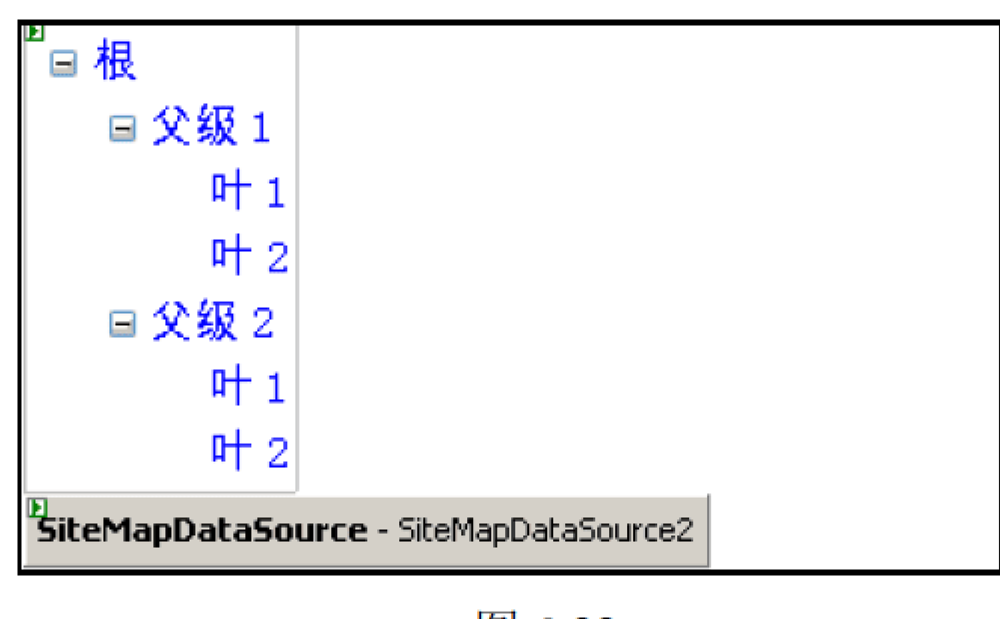


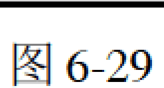
```
        <siteMapNode title="计算机书籍" url="sitemapsA.aspx">
            <siteMapNode title="编程语言" url="sitemapsA1.aspx" />
        </siteMapNode>
        <siteMapNode title="人文类书籍" url="sitemapsB.aspx">
            <siteMapNode title="历史" url="sitemapsB4.aspx">
                <siteMapNode title="近代史" url="sitemapsB4a.aspx" />
                <siteMapNode title="现代史" url="sitemapsB4b.aspx" />
            </siteMapNode>
        </siteMapNode>
    </siteMapNode>
</siteMap>
```

以上的 Web.sitemap 文件，包含一个根节点和多个嵌套节点，并且还为每个节点设置了 title 和 url 属性。在 TreeView 控件中，title 属性将显示为节点名称，url 属性将作为节点的超链接地址。

(5) 打开页面 Default.aspx，在页面中添加 TreeView 控件和 SiteMapDataSource 控件，设计界面如图 6-29 所示。

(6) 点击 TreeView 控件的智能标记，出现如图 6-30 所示的对话框。




图 6-29

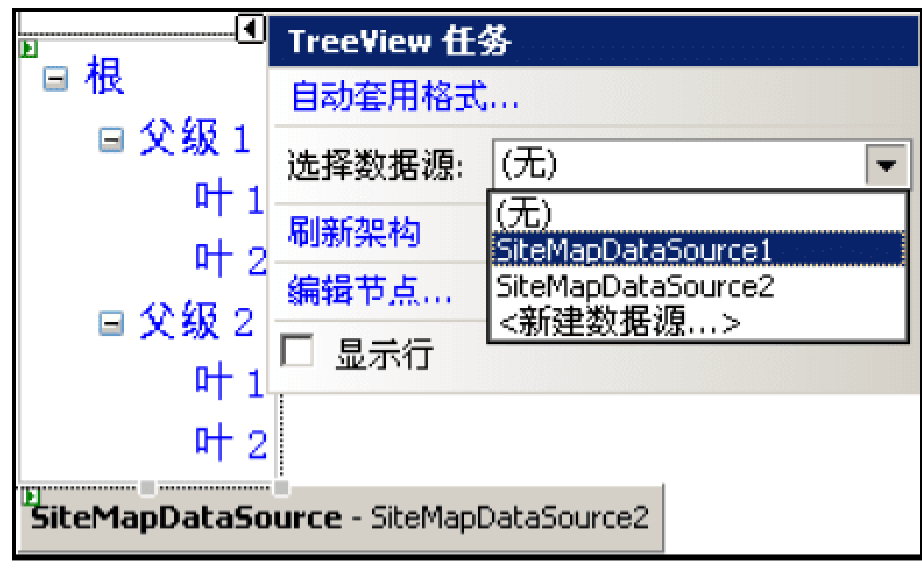


图 6-30

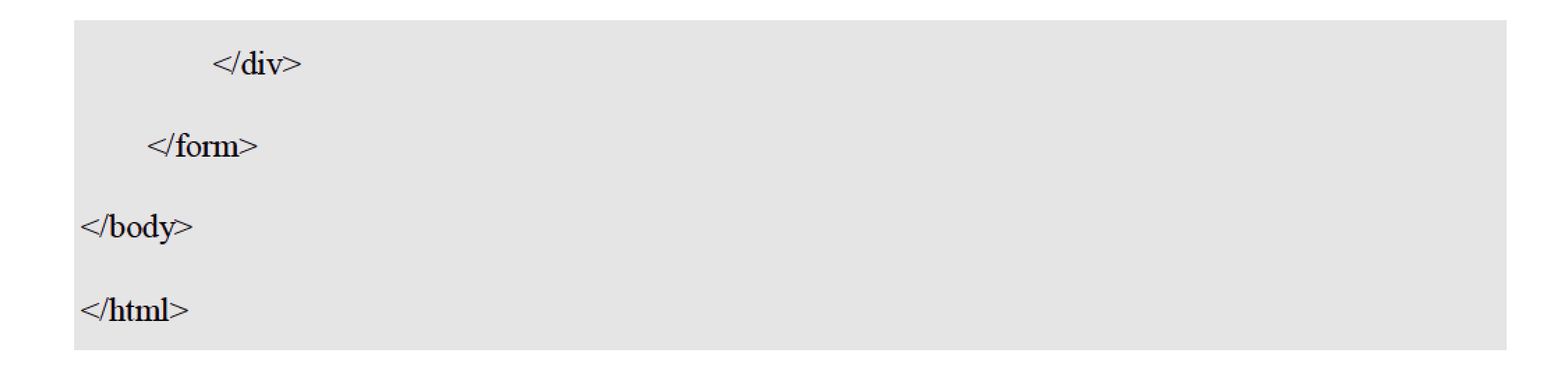
(7) 在对话框中的“选择数据源”下拉框中，选择添加的 SiteMapDataSource 控件。这时 SiteMapDataSource 控件默认处理 Web.sitemap 文件。选择后，效果如图 6-31 所示。

图 6-31

(8) Default.aspx 文件的源代码如下所示:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
<link id="InstanceStyle" href="StyleSheet.css" rel="stylesheet" />
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <fieldset style="width: 240px">
                <legend class="mainTitle">实现 TreeView 控件绑定 Web.sitemap 文件
                          </legend><br />
                <asp:TreeView ID="TreeView1" runat="server"
                             DataSourceID="SiteMapDataSource1">
                </asp:TreeView>
                <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
            </fieldset>
             
```





```
        </div>
    </form>
</body>
</html>
```

如上所示粗体代码，包含 TreeView 控件和 SiteMapDataSource 控件。代码 DataSourceID="SiteMapDataSource1"，说明了 TreeView 控件与 SiteMapDataSource 控件之间的数据绑定关系。

练习 2：为 eshop 电子商务网站创建客户母版页 User.master。

【问题描述】

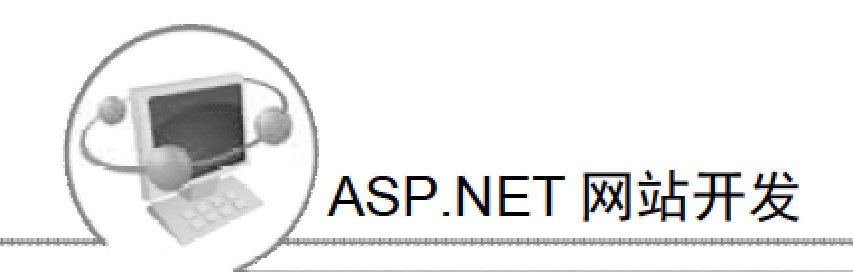
所有客户共用同一个母版页，在该母版页中可以超链接到和客户相关的页面，效果如图 6-32 所示。



图 6-32

【问题分析】

头和尾可以使用用户控件 head.ascx 和 foot.ascx；左侧添加 TreeView 控件，右侧的 fieldset 的标题 legend 中放置一个 ContentPlaceHolder，内容部分再放置一个 Content PlaceHolder。



【课后作业】

为 eshop 电子商务网站创建管理员母版页 manage.master，界面如图 6-33 所示。

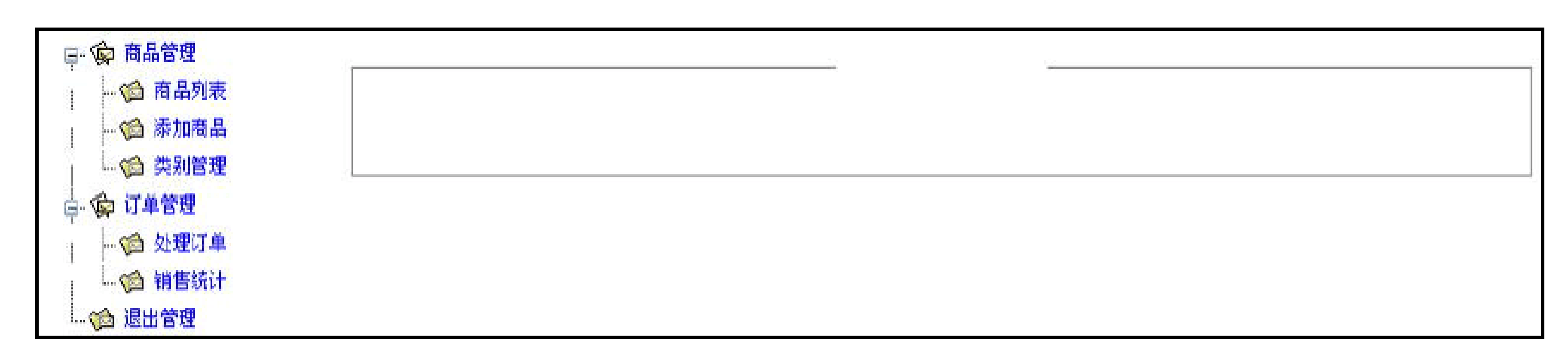


图 6-33

第7章

项目整合和主题



课程目标

- 了解网站开发的一般步骤
- 创建主题
- 应用主题



简 介

软件开发是一个系统工程，它需要很多部门和人员的参与和配合，在这一章中我们将向大家介绍程序员与美工在项目开发中各自的工作及相互配合。

站点外观主要与页面和控件的样式属性有关。几乎所有控件都支持 Style 对象模型(包括一些扩展样式属性)，其可用于设置字体、边框、背景色、前景色、高度和宽度等样式属性。同时，控件还完全支持将样式设置与控件属性分离的级联样式表(CSS)。在实现站点过程中，开发人员可能不得不为多数控件添加样式属性，这种做法不仅繁琐，而且不容易保持站点外观的一致性和独立性。理想的方法是：只要为控件定义一次样式属性，就能够方便地应用到站点的所有页面中。为了实现这个方法，ASP.NET 3.5 提供了一种称为“主题”的功能。本节将对主题功能进行概括性说明，内容包括组成元素、文件存储和组织方式、全局主题和应用程序主题等。

7.1 项目整合

7.1.1 网站开发步骤

开发一个网站从开始立项到最终交付给用户使用，会经历很多的步骤。无论网站大小、简单与复杂，大抵都会经历相同的阶段。下面我们就模拟一个五人组成的小团队，为某企业开发一个商用网站的场景，给大家讲一下网站开发所要经历的步骤。

首先介绍团队中应有的角色：

- 项目组长：组织、分配、管理者。
- 界面工程师：是用户界面交互方面的专家，决定与用户交互的方式，当然很大程度上也影响着界面。
- 美工：设计和美化界面。



- 高级程序员：设计总体程序结构，制定技术上的规范，并为小组解决各种难题，帮助项目组长分解每日程序员任务。
- 程序员：编写代码，实现功能。
- 需求人员：需求分析、设计。
- 其他：如测试人员、文档管理人员等。

大致的工作流程如下：

(1) 公关人员和需求人员获得用户需求，并制定需求文档。

需求的正确与否是项目成功的首要关键环节，他们需要获取到用户的各种习惯，主要分为两种思路来整理，一种是之前用过软件系统的考虑如何延续他们的习惯，另一种是之前没有用过软件系统的考虑如何适应他们原有手工的工作流程，并做出合理化的改进。

(2) 项目组长和需求人员以及高级程序员共同根据需求制定大体的设计方案，包括总体模块和各个可行性功能。

在这里，项目组长会根据需求人员和高级程序员的意见来合理安排出一个基本雏形，然后去写 Project(项目管理软件)……后面还有反复交付雏形给用户确认等。有一点值得注意的是，项目组长除了需要具备一定的人员管理方法以外，最好还是要懂得技术，这样能够制定出更合理、更准确的项目进度，也能带动团队工作的士气。

(3) 开工，项目组长在高级程序员配合下根据预先计划开始推动项目进展。

首先由项目组长和高级程序员在上一环节设计的雏形的基础上按照计划规划架设各模块的基本结构。然后以模块为单位，一般团队喜欢采用称为狼群战术的方法来逐步蚕食各个模块。每个模块的流程分为如下几个步骤：

- 高级程序员详细拆分该模块的各个界面和功能，包括前台和后台等。需要需求人员给出参考。
- 在高级程序员的分配下，界面工程师对当前子模块制定界面用户交互的基本方案，也需要需求人员给出参考，美工人员则给出美学方面的建议，并达成一致。在这里，界面工程师会将决定界面的大致框架，并将界面相应的功能描述成文以用于给程序员。一个子模块界面的雏形在这里已经诞生，生成的程序文件有 aspx 和 cs。

- 美工去做界面，对界面工程师所搭建的界面框架 aspx 或 ascx 文件进行处理，如背景、需要配合的图片图标及 flash 等。在这个环节上，美工和界面工程师已经在明确需求人员的指导下达成对界面统一风格的一致。因为界面工程师在之前已经在页面中制定好标记，所以美工可以忽略有脚本标记的地方。而且，总的来说这一环节上美工主要是预先为界面定义好各种素材。
- 与美工并发执行的是高级程序员与程序员对功能的实现。程序员们在界面工程师的指导下将功能实现，其间包括满足交互功能所需的控件、业务规则层、数据访问层等等的实现，所涉及编写的文件则为界面文件(ascx 等)和程序文件 cs。这里需要说明的是在实现功能时程序员只要把满足功能的控件拖到大致位置即可，然后关注功能的实现。而此时美工也在设计该界面，但因为只是设计素材，所以根本不与程序员冲突，在后面的环节中始终以程序员完成的程序文档为准。
- 程序员完成功能后，转交测试人员进行功能测试。
- 基本测试通过后，又回到界面工程师手里，在不改动程序文件(cs)的前提下，界面工程师只对界面文件中的各种控件、结构等进行调整，直至达到满意的效果。
- 界面基本已经诞生，只是不太靓丽，所以这时回到美工手上，给其穿上美工设计的“靓装”，加上各种图片背景等就可以了。
- 项目组长贯穿整个过程，负责团队人员之间的协调，监督项目进度，合理分配任务。

(4) 所有模块都完工后，就是整体的衔接和测试，然后反复交付用户征求意见。这里参与的是团队所有的人马，一直工作到最后期限为止，然后再延期，直到用户满意。

7.1.2　程序员与美工

现在公司的项目一般都是基于 B/S 结构的，绝大多数操作界面都是通过网页的形式展现在用户面前的，页面的美观就成了非常重要的问题，同时涉及的就是程序员与美工的合作问题。其中主要的问题、矛盾有以下几点：

- 美工何时参与到项目中来。



- 程序员不懂如何将页面做得美观，美工也不懂如何向页面中添加代码。
- 程序员和美工是两种完全不同的人，他们关心的事情也完全不同，这就导致两种人对页面代码(html)风格的要求大相径庭——程序员要的是简单易懂，美工要的是美观漂亮。
- 程序员要做的是将数据展现在页面上(使用简单的条件、循环语句)，美工要做的是将美丽的界面充满整个屏幕。

上面的这几点问题和矛盾从关系上来讲是层层递进的，要一个一个依次解决。先来说说美工何时介入到项目中来，大致有以下 3 种：①先由美工制作静态页面，完成后由程序员直接向页面中添加程序代码；②程序员随时和美工沟通，向美工描述页面需求，随要随做；③程序员自己编写测试页面，然后让美工进行美化。

这 3 种方式可以说各有利弊。方式一对程序员来说绝对是个好方式，它能使程序员最大限度地远离那些繁琐的页面编码，提高程序员工作的效率。同时，一套完整的页面可以展现全部业务的流程，对程序员开发也起到了规范的作用。但这种方式对美工的要求极高，美工要了解项目的需求，而这一般是达不到的。但可以让了解需求的人为其讲解，或是描绘出希望的页面的样式。这样虽然可以弥补美工对业务了解的不足，但也确实花掉了重量级程序员的很多时间。方式二是一个比较折中的方法，这样做无须太多的准备就可开始编码工作，程序员把握页面内容和样式，向美工详细描述，美工再根据描述设计页面，最后返回给程序员添加代码。这个反馈的过程一般比较迅速，效果也不错，可以达到程序员预期的效果，适用于项目时间要求比较紧的情况。该方式的问题在于没有一个形象化的完整的流程可供程序员参考，一切掌握在程序员手中，容易造成系统整体风格的不统一。方式三一般用于对已有项目的美化上，对美工的要求也很高，他们需要具备在 html 和其他代码混合体的环境下工作的能力，而且修改的效果一般不是很佳，不到万不得已不推荐使用。

上述方式虽然表现出来的问题各不相同，但解决的方法却很相似。首先，美工要养成一些程序员编码时惯有的习惯，例如：文件命名要有意义、html 代码要根据层次进行缩进等。其次，页面代码的一些细节也要注意，例如，使用居中或右对齐标签来取代空格，必须使用空格时也要用“ ”，不使用<p>标签，尽量使用表格等。再次，如果在条件允许的

情况下，美工也可以学习一下夹杂在页面中的各种程序代码，了解其语义和工作原理，这将对与程序员的合作起到很大的帮助。最后，就是程序员要在向页面文件中添加代码前先对页面代码做一下审核工作，在这里并不是看美工的页面是否美观，而是看在原有页面代码的基础上是否能够使用简单的条件、循环语句来显示数据(例如，页面布局过于复杂，不能通过简单的循环来显示所有数据)，否则就需要修改页面代码直到能满足要求为止。

做网站后台的一般流程如下。

1. 网站规划阶段

这个阶段主要是对网站的功能、目标受众、内容、栏目进行规划。这期间会经常性地和有关领导进行沟通。首先，自己一定要对网站的整体规划清清楚楚，然后要吸取别人的建议。项目的大致进度，要在这个阶段结束的时候确定下来。

2. 后台模块划分和版面设计

这个阶段，程序员要和美工兵分两路，分头行动。后台模块划分如果做好了，后面的效率就会高一些，这个过程不能省。版面设计，美工既要考虑网站整体规划，又要考虑大家的建议，尤其是不能忽视领导们的观点(虽然大多数情况下领导的美术细胞少得可怜)。在这个大前提下，再兼顾美观、合理。一个好的美工，不仅能做出漂亮的页面，还要能迎合一下客户或者公司领导的意愿，而且能和程序员进行沟通。

在这个阶段，程序员和项目经理(项目负责人)要拿出一个可操作的模块划分方案，而美工要确定网站的版面框架、美术风格，做出网站首页和二级页面。实际上，在第一个阶段(网站规划阶段)，美工就应该开始思考网站的风格了。在第二个阶段，则需要把比较抽象的初级设想变成具体的页面。基本上，首页定了，整个网站的页面就定了一大半。在这个阶段结束的时候，要将项目的进度计划进一步具体化。

3. 数据库设计

这项工作很重要，程序员应该知道怎么去做。而且数据库设计是和一个人的理论水平、实际经验息息相关的。大的、复杂的站点，数据库规划可能要用一周左右的时间，小的、简单的站点，数据库设计也需要 2 到 3 天。



在这个阶段，美工也没有闲着，继续完成页面设计。要知道下一个阶段，程序员可就要用到美工的页面了。最好别出现这样的情况：程序员要用到某个页面，而美工还没有把那个页面确定下来。

4. 后台程序编码

这个阶段，程序员要紧张工作，会比较辛苦。程序员需要遵守的三个原则：团队合作、保证进度、保证质量。美工这个时候要辅助程序员做页面。这个阶段美工可能比较闲，但是一定要称职。

5. 除错、改进、页面美化

这个阶段，程序员需要细心和耐心，除错是一个细活，需要程序员通过调试发现潜在的 BUG。在较成熟的开发团队中，找到 BUG 的事是由测试工程师来完成的。改进发现的 BUG 是完善程序不可少的一步。最后由美工对页面进行美化达到最终想要的效果。

7.2　创建主题

本节将围绕如何创建主题这一问题展开讲解，内容包括创建皮肤文件、为主题添加 CSS 文件、在主题中使用图片等。

7.2.1　创建皮肤文件

皮肤文件是主题的核心内容。在介绍创建皮肤文件的方法之前，必须了解与创建皮肤文件有关的知识。

1. 控件皮肤类型

每个皮肤文件中都可以定义一个或者多个控件皮肤设置。根据应用情况，控件皮肤可分为两种类型：默认皮肤和命名皮肤。当向页面应用主题时，默认皮肤将自动应用于同一类型的所有控件。如果控件皮肤没有包含 SkinID 属性，则是默认皮肤。例如，为 Button

控件设置的皮肤就是一个默认皮肤。页面中所有的 Button 控件都将应用该皮肤定义的属性设置，然而，Button 控件皮肤设置仅适用于 Button 控件，而不适用于 LinkButton 控件或从 Button 对象派生的控件。需要注意的是，针对一种类型控件，仅能设置一个默认皮肤。命名皮肤是设置了 SkinID 属性的控件皮肤，其不会自动按类型应用于控件，而是通过设置控件的 SkinID 属性，将命名皮肤显式应用于控件。通过创建和应用命名皮肤，可以为同一类型控件的不同实例设置不同的皮肤。在创建和应用命名皮肤时，可以为同一类型控件的不同实例设置不同的皮肤。在创建控件皮肤的过程中，可以为一种类型控件设置多个命名皮肤(SkinID 属性值不能重复)。

2. 控件皮肤设置的属性要求

控件皮肤只能针对具有 Themeable(可设置主题)元数据的控件属性进行设置。通常情况下，多数服务器控件都支持主题功能，但是也有部分控件不支持。例如，数据源控件就是其中之一。如果要禁用针对某个控件的主题应用，那么只要将其 EnableTheming 属性值设置为 false 即可。任何控件的 ID 属性都不可在控件皮肤设置中出现。如果向皮肤文件中添加了不能设置主题的属性，将会导致错误发生。

3. 控件皮肤设置的属性类型

控件皮肤设置的属性可以是简单属性，也可以是复杂属性。简单属性是控件皮肤设置中最常见的类型，例如，BackColor、Width 等。复杂属性主要包括集合属性、模板属性、数据绑定表达式等类型。

下面列举了一个简单的创建皮肤文件的示例。该示例中包括 3 个 TextBox 控件，其中一个设置了默认皮肤，另两个设置了命名皮肤。图 7-1 所示显示了示例效果图。

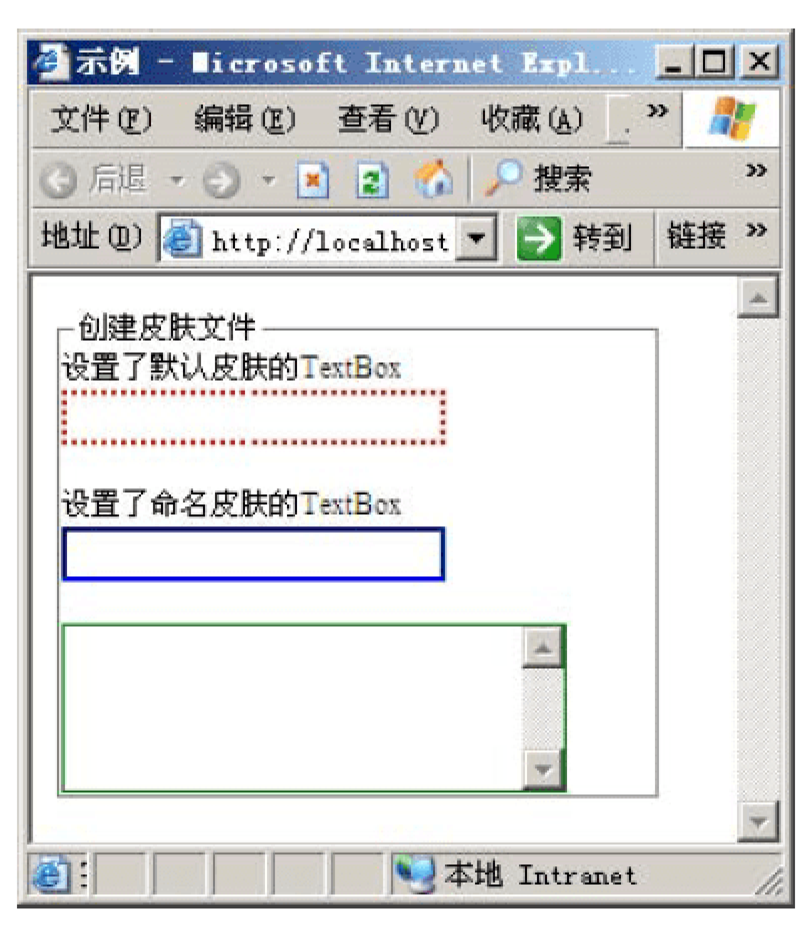
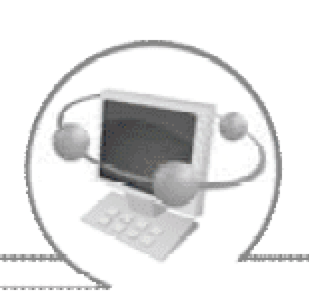


图 7-1

图中由上至下，第一个 TextBox 控件是默认皮肤设置的外观，后两个 TextBox 控件是命名皮肤设置的外观。



以上应用程序包括两个核心文件，一个是皮肤文件 TextBox.skin，另一个是 ASP.NET 应用程序文件 Default.aspx。下面先介绍 TextBox.skin 文件的创建方法。

为了创建皮肤文件，必须在应用程序根目录下创建一个名为 App_Themes 的文件夹用于存储主题，然后，在该文件夹下创建一个名为 mytheme 的子文件夹，表示所创建的主题名为 mytheme。最后，在 mytheme 文件夹下创建一个 TextBox.skin 文件。TextBox.skin 就是皮肤文件，其中包含了对 TextBox 控件的皮肤设置。下面列举了 TextBox.skin 文件的源代码。

```
<asp:TextBox runat="server" BorderColor="red" BackColor="white" BorderStyle="dotted" />
<asp:TextBox runat="server" BorderColor="blue" BackColor="white" SkinID="Blue" />
<asp:TextBox runat="server" BorderColor="green" BackColor="white" Width="200px"
BorderWidth="1px" SkinID="Green" />
```

如上代码所示，针对 TextBox 控件共设置了 3 个皮肤。其中，没有添加 SkinID 属性的是 TextBox 的默认皮肤，另外两个设置了 SkinID 属性的是 TextBox 的命名皮肤。它们的 SkinID 属性值分别为 Blue 和 Green。至此完成了创建控件皮肤文件的过程。

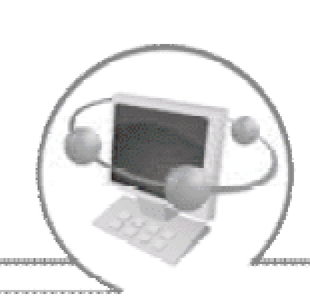
实际上，可以发现一个创建控件皮肤的简单方法。首先将控件添加到.aspx 页面中，然后，利用 Visual Studio 2008 的属性窗口及可视化设计功能对控件进行配置，这样控件就具有了所需外观，最后，再将控件代码复制到皮肤文件中并做适当修改即可。例如在本例中，可将 TextBox 控件添加到页面中，并设置边框颜色、背景颜色等属性。然后，将控件定义代码复制到皮肤文件中。注意别忘记移除皮肤文件中控件的 ID 属性。通过以上设置，就可以轻松创建控件的默认皮肤。如果还添加了 SkinID 设置，那么创建的则是命名皮肤。

创建 Default.aspx 文件的主要目的是应用 TextBox.skin 文件中的控件皮肤设置。下面列举了 Default.aspx 文件的源代码。

```
<%@ Page Language="C#" AutoEventWireup="true" Theme="mytheme" CodeFile="Default.aspx.cs"
Inherits="_Default" %>
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>示例</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <fieldset style="width: 240px">
                <legend>创建皮肤文件</legend>
设置了默认皮肤的 TextBox
<asp:TextBox runat="server" ID="TextBox1" />
<br /><br />
设置了命名皮肤的 TextBox
<asp:TextBox runat="server" ID="TextBox2" SkinID="Blue" />
<br /><br />
<asp:TextBox runat="server" ID="TextBox3" SkinID="Green" TextMode="MultiLine"
Rows="4"></asp:TextBox>
            </fieldset>
        </div>
    </form>
</body>
</html>
```

如上代码所示，Default.aspx 文件中主要包括了 3 个 TextBox，分别应用了 TextBox.skin 文件中设置的 3 个有关 TextBox 控件的皮肤。应用主题的方法是在<%@ Page %>标签中设置一个 Theme 属性。例如，TextBox.skin 文件位于主题 mytheme 中，因此，在 Default.aspx 中应用 mytheme 主题时，必须将 Theme 属性值设置为 mytheme。此时，TextBox.skin 中设



置的内容都将自动应用到 Default.aspx 上。例如，ID 为 TextBox1 的 TextBox 控件应用了 TextBox.skin 中定义的默认皮肤；ID 为 TextBox2，SkinID 为 Blue 的 TextBox 控件应用了 TextBox.skin 中定义的 SkinID 为 Blue 的命名皮肤；ID 为 TextBox3，SkinID 为 Green 的 TextBox 控件应用了 TextBox.skin 中定义的 SkinID 为 Green 的命名皮肤。由此可见，SkinID 属性是创建和应用控件命名皮肤设置的关键。另外，在 ID 为 TextBox3，SkinID 为 Green 的 TextBox 控件设置中，还定义了 TextMode 和 Rows 属性，并且这两个属性也随同命名皮肤共同作用于控件。这说明控件皮肤的设置并不会独占控件，控件皮肤和页中的控件属性设置将合并，以构成控件的最终设置。本例中，控件代码添加了与控件皮肤不同的属性设置，例如，TextMode 和 Rows 属性都没有在皮肤文件中出现。如果在控件代码中添加了与控件皮肤相同的属性设置，例如，设置 Width= " 400px " ，那么页面将首先应用控件属性设置，然后应用主题，以重写控件属性设置。页面最终显示的是控件皮肤的设置效果。

7.2.2　为主题添加 CSS 文件

主题中的 CSS 文件主要用于设置页面和普通 HTML 控件的外观样式。例如，设置页背景色、普通文本的字体等。在未出现主题功能之前，必须在页面中引用 CSS 文件链接，才能加载所设置的内容。在 ASP.NET 2.0 提供了主题功能后，这种情况得到改观。开发人员可以在一个主题文件夹中添加一个或者多个 CSS 文件，这些 CSS 文件无须引用链接，就能够自动随主题应用到页面中。需要注意的是，在页面头部必须定义<head runat= " server " /> 标记。如果页面既包含主题应用(主题中包含 CSS 文件)，又引用了 CSS 文件链接，那么两者将共同作用于页面。下面通过一个简单的示例，说明主题添加 CSS 文件的方法。

本例实现的内容包括为页面背景色、页面中所有超链接文本和普通 HTML 文本框创建样式。如图 7-2 所示显示了示例效果。

页面中共有 3 处被设置了样式。一是页面背景色；二是超文本的外观、悬停效果；三是普通 HTML 文本框的背景色。

在实现以上示例之前需要明确：无论是页面背景，还是超链接文本和 HTML 文本框，都是页面和普通的 HTML 控件，不是服务器控件。因此，不能使用控件皮肤对其进行外观

设置，只能使用 CSS。为此，需要在主题文件夹 mytheme 中，创建一个 CSS 文件 StyleSheet.css，该文件的源代码如下所示。

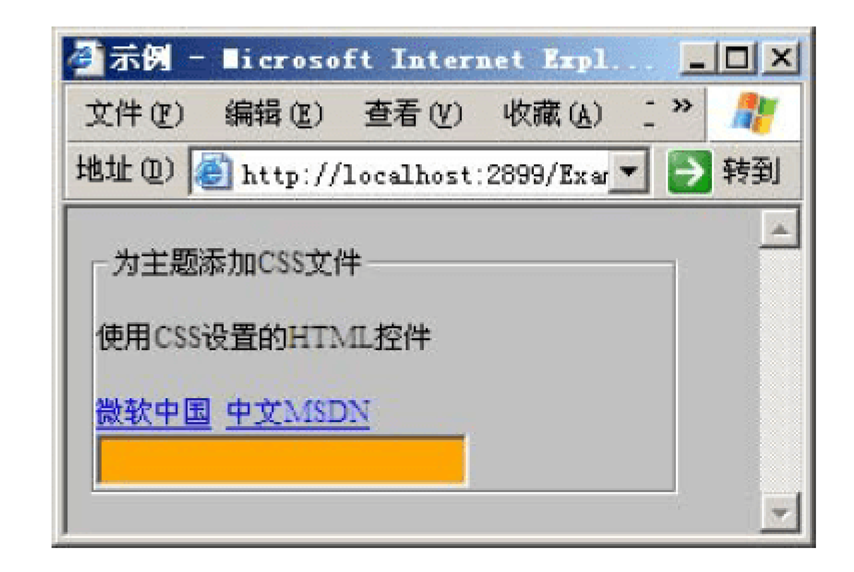


图 7-2

```
body
{
    background-color: silver;
}
A:link
{
    color: Blue;
    text-decoration: underline;
}
A:visited
{
    color: Blue;
    text-decoration: underline;
}
A:hover
{
    color: Red;
    text-decoration: none;
```



```
}
INPUT
{
    background-color: Orange;
}
```

以上代码很简单，其中包括 5 个样式表元素，它们是用于设置页面的元素 body、用于设置超链接文件的 A.link、A.visited、A.hover，以及设置输入控件的元素 INPUT。通过为这些元素添加样式属性，能够实现对相关内容的外观定制。由此可见，主题中包含的 CSS 文件与普通的 CSS 文件没有任何区别。需要再次强调的是，主题中的 CSS 文件主要针对页面和普通 HTML 控件进行设置，该文件必须保存在主题文件夹中。

下面是 Default.aspx 文件源代码：

```
<%@ Page Language="C#" AutoEventWireup="true" Theme="mytheme" CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>示例</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <fieldset style="width: 240px">
<legend>为主题添加 CSS 文件</legend>
```

```
<br />
使用 CSS 设置的 HTML 控件<br /><br />
<a href="http://www.microsoft.com/china/" target="_blank">微软中国</a>
<a href="http://www.microsoft.com/china/msdn/" target="_blank">
中文 MSDN</a><br />
<input type="text" /><br />
            </fieldset>
        </div>
    </form>
</body>
</html>
```

以上代码很简单，无须过多说明。使用主题中的 CSS 文件，至少应该确保两点。一是页面必须应用对应主题，例如，上文代码中设置了 Theme="mytheme"，这使得 Default.aspx 页面应用主题 mytheme。二是必须保证在页面头部定义<head runat="server" />标记，否则，主题中的 CSS 文件将无法正常应用。

7.2.3　在主题中使用图片

主题中可以包括图片，通常是皮肤文件中控件皮肤设置所引用的图片。例如，当为 Menu、TreeView、BulletedList 控件设置皮肤时，可能引用图片。通常情况下，图片文件应存储在主题文件夹的图片子文件夹中。同时，控件皮肤设置引用图片时，建议使用相对路径，以使整个主题可以方便地移植到其他应用程序中。

下面列举了一个为 BulletedList 控件设置包含图片的皮肤的示例。通过该示例，可以掌握在主题中使用图片的实现方法。图 7-3 所示显示了示例效果。

页面中包含了一个 BulletedList 控件，该控件用于显示列表项内容。每个列表项不仅包括文字内容，而且在文字左侧都设置了一个小图片，该图片名为 arraw.gif。





图 7-3

实现以上示例关键要设置好两项内容。一是设置 BulletedList 控件的皮肤，二是设置 BulletedList 控件的列表项。很显然，控件皮肤的设置必须包含在皮肤文件中。而对于设置列表项则有两种选项。既可以将列表项内容设置在 ASP.NET 页面的控件声明中，也可以设置在控件皮肤中。本例首先使用前一种方法实现，然后，再说明后一种方法。为了便于主题移植到其他应用程序中，还必须在主题文件夹 mytheme 中创建一个子文件夹 Images，该文件夹中包含列表项文字旁的图片 arraw.gif。

下面列举了皮肤文件 BulletedList.skin 文件源代码：

```
<asp:BulletedList BulletStyle="CustomImage" Runat="Server"
BulletImageUrl="Images/arrow.gif" />
```

如上代码所示，BulletedList 控件皮肤主要包括对 BulletStyle 和 BulletImageUrl 属性的设置。属性 BulletStyle 用于设置列表项编号外观。当该属性值设置为 CustomImage 时，表示使用自定义图片作为列表项编号。属性 BulletImageUrl 用于设置自定义图片的路径。本例中，自定义图片路径使用了图片相对于皮肤文件的路径。

Default.aspx 文件源代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true" Theme="mytheme" CodeFile="Default.aspx.cs"
Inherits="_Default" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
```

```
<title>示例</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <fieldset style="width: 240px">
                <legend>在主题中使用图片</legend>
                <asp:BulletedList ID="BulletedList1" runat="server">
                    <asp:ListItem Value="0">JSP 程序设计</asp:ListItem>
                    <asp:ListItem Value="1">计算机网络教程</asp:ListItem>
                    <asp:ListItem Value="2">参考手册</asp:ListItem>
                    <asp:ListItem Value="3">UML 和模式应用</asp:ListItem>
                </asp:BulletedList>
            </fieldset>
        </div>
    </form>
</body>
</html>
```

如上代码所示，Default.aspx 文件应用了主题 mytheme。同时，页面中主要包括了一个 BulletedList 控件，该控件中包括 4 个列表项。BulletedList.skin 中设置的 BulletedList 控件皮肤将自动应用到页面上。

前文已经说明，皮肤文件中可以包含集合属性，因此，本例可以将 BulletedList 控件中多个选项设置定义在皮肤文件中，示例代码如下所示：

```
<asp:BulletedList skinid="WithItem" BulletStyle="CustomImage"
BulletImageUrl="Image/arrow.gif" Runat="Server">
```



```
<asp:ListItem value="0">JSP 程序设计 1</asp:ListItem>
<asp:ListItem value="1">计算机网络编程 2</asp:ListItem>
<asp:ListItem value="2">参考手册 3</asp:ListItem>
<asp:ListItem value="3">UML 和模式应用 4</asp:ListItem>
</asp:BulletedList>
```

7.3　应用主题

上文介绍的示例中简单说明了应用主题的方法，即在页头<%@ Page %>中设置 Theme 属性值为主题名。实际上，主题的应用并没有那样简单。本节将详细介绍与主题应用有关的内容，包括指定和禁用主题、动态加载主题等。

7.3.1　指定和禁用主题

前文示例中的主题主要应用于单个页面范围。当开发人员在页头设置 Theme 属性值之后，整个页面内容都将自动应用所设置的主题。按照这种实现方法，如果整个应用程序，整个服务器都需要设置主题，那么岂不是要为每个页面都设置 Theme 属性吗？这未免过于繁琐了。本节将介绍为页面和应用程序指定和禁用主题的实现方法。

1. 为页面指定和禁用主题

通常而言，为单个页面指定主题有两种方法。一种是前文介绍的设置 Theme 属性，另一种是设置 StyleSheetTheme 属性。本小节重点说明使用 StyleSheetTheme 设置主题的方法。

当主题应用一页面时，主题中所设置的控件皮肤优先于页面中直接设置的任何控件属性。例如，如果主题指定所有 TextBox 控件的背景都显示为银色，那么即使在页面中为个别 TextBox 控件设置了不同的 BackColor 属性值，页面中所有 TextBox 控件的背景色仍然都显示为银色。在某些情况下，开发人员可能希望替代页面中某个 TextBox 的主题设置。例如，当页面中存在多个 TextBox 控件时，可能会希望将某个 TextBox 控件的 BackColor

属性更改为红色，以便突出显示该控件。这种情况下，需要使用 StyleSheetTheme。

StyleSheetTheme 的工作和应用方式与普通主题(使用 Theme 设置的主题)相似。只不过当使用 StyleSheetTheme 时，控件皮肤的设置可以被页面中声明的同一类型控件的相同属性所替代。下面列举了一个 TextBox.skin 文件源代码，其中包含一个 TextBox 控件皮肤设置。该设置将每个 TextBox 控件的 BackColor 属性设置成了银色，将 ForeColor 设置成了蓝色。

```
<asp:TextBox BackColor="Silver" ForeColor="Blue" Runat="Server" />
```

假设以上皮肤添加到了主题 mytheme 中，那么使用 StyleSheetTheme 属性来应用该主题的 Default.aspx 文件源代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true" StyleSheetTheme="mytheme"
CodeFile="Default.aspx.cs" Inherits="_Default" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>示例</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <fieldset style="width: 240px">
                <legend>使用 StyleSheetTheme 属性</legend>
                设置了默认皮肤的 TextBox
                <asp:TextBox runat="server" ID="TextBox1"></asp:TextBox><br />
                <br />
                设置了 BackColor 属性的 TextBox
                <asp:TextBox runat="server" ID="TextBox2"
```



```
                    BackColor="Red"></asp:TextBox><br />
                </fieldset>
            </div>
        </form>
</body>
</html>
```

如上代码所示，该页面通过 StyleSheetTheme 属性设置使用 mytheme 主题，而没有使用 Theme 属性进行设置。同时，代码中第二个 TextBox 控件的声明代码还设置了一个 BackColor 属性值为 Red。根据上文介绍的内容，第二个 TextBox 控件的 BackColor 将代替 TextBox.skin 中设置的 BackColor 属性，使其背景显示为红色。第一个 TextBox 控件则使用控件皮肤的默认设置，背景显示为银色。如图 7-4 所示显示了示例效果。

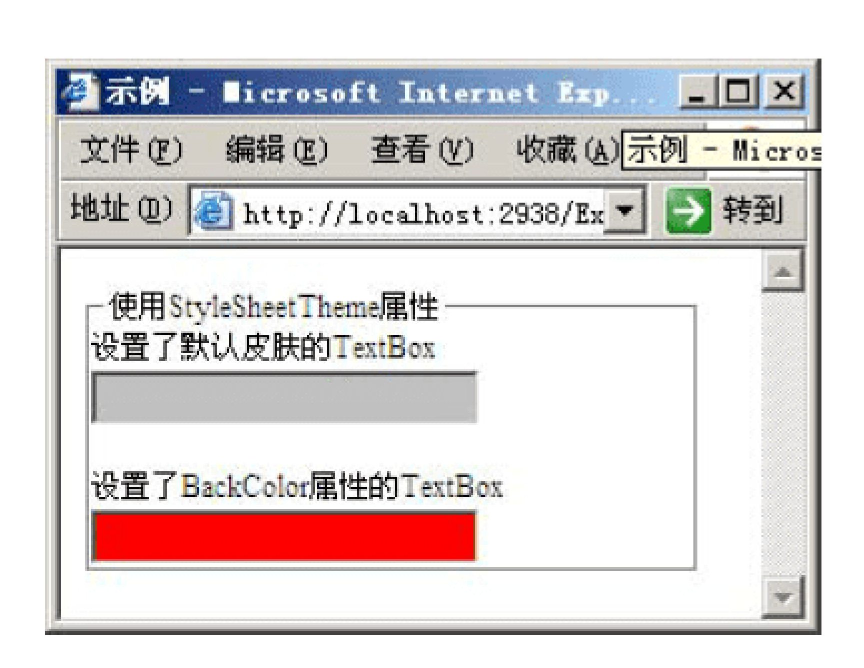


图 7-4

我们是否能够在页面中同时设置Theme 和 StyleSheetTheme 属性呢？如果可以，Theme、StyleSheetTheme 和控件声明属性所设置的内容如何显示呢？可以肯定的是，在一个页头的 <%@ Page %>中，允许同时设置 Theme 和 StyleSheetTheme 属性，并且两者的属性值可以不相同。然而，在应用过程中，必须了解两者同时使用时对控件应用的优先级顺序。首先，应用的是 StyleSheetTheme 属性的设置，然后，应用页面中的控件声明代码所设置的属性，如果这些属性与 StyleSheetTheme 的设置属性重复，则替代 SytleSheetTheme 的设置，最后，应用 Theme 属性的设置内容，如果出现属性重复的情况，Theme 属性设置将重写控件声明属性和 StyleSheetTheme。

以上介绍了为页面指定主题的方法。另外，还可以禁用页面主题，其实现方法很简单。只要在页头<%@ Page %>中设置 EnableTheming ="false"即可。

2. 为应用程序指定和禁用主题

每个应用程序中都包括多个页面，并且为了保证和谐统一的用户界面，开发人员往往希望所有页面使用同一主题。如果为在每个页面头都设置相同的 Theme 属性值，那么必将繁锁。为了快速地为整个应用程序的所有页面设置相同的主题，可以设置 Web.config 文件的<pages>配置节内容。下面列举了 Web.config 文件的配置示例源代码：

```
<configuration>
    <system.web>
        <pages Theme="主题名" />
    </system.web>
</configuration>
```

当 Web.config 文件完成如上配置后，整个应用程序将自动应用所设置的主题，而不需要为每个页头设置 Theme 属性。Web.config 文件中的主题设置会应用于该应用程序中的所有 ASP.NET 网页。如果需要，也可以将<pages>配置节中的 Theme 属性更改为 SthleSheet Theme 属性。需要注意的是，同一应用程序可以包含用于指定主题的多个 Web.config 文件。可以将不同的 Web.config 文件添加到不同的子文件夹中，每个 Web 配置文件都可以指定不同的主题。

如果需要禁用整个应用程序的主题设置，只要将<pages>配置节中的 Theme 或者 StyleSheetTheme 属性值设置为空即可。

7.3.2 动态加载主题

动态加载主题是指通过编程方式，在运行时动态修改页面所应用的主题，从而使用户摆脱设置主题的限制，实现动态自定义页面主题的功能。实现动态加载主题的核心是修改 Page 对象的 Theme 属性值。可以将任何有效的主题名指派给该属性。然而，在实现过程中



需要注意：Theme 属性只能在页面的 PreInit 事件发生过程中或者之前设置(多数是在 PreInit 事件中)。这意味着必须在 Page_PreInit 事件处理程序中修改 Theme 属性值。

下面通过一个示例来说明动态加载主题的实现方法。图 7-5 和图 7-6 显示了示例效果。



图 7-5



图 7-6

页面中包括一个下拉框控件和一个日历控件。下拉框控件中包含两个选项，一个是“启用 Red 主题”，另一个是“启用 Blue 主题”。默认情况下，页面加载如图 7-5 所示的 Red 主题。当更改了下拉框选项后，页面将自动刷新，并加载选中项所指示的主题。

实现以上示例关键要创建两个主题和一个应用主题的 Default.aspx 文件。两个主题分别设置在主题文件夹 Blue 和 Red 中，并且每个主题中包含一个皮肤文件，用于设置日历控件的皮肤。下面列举了位于 Red 主题文件夹中的 Calendar.skin 文件源代码。

```
<asp:Calendar runat="server" BackColor="White" BorderColor="#EFE6F7"
    CellPadding="4" DayNameFormat="Shortest" Font-Size="0.8em"
    ForeColor="Black" Height="180px" Width="200px">
<SelectedDayStyle BackColor="#8A170F" Font-Bold="True" ForeColor="White"
     Font-Size="0.8em"/>
    <SelectorStyle BackColor="#8A170F" Font-Size="0.8em"/>
    <WeekendDayStyle BackColor="#E7E7E7" Font-Size="0.8em"/>
    <OtherMonthDayStyle ForeColor="#8A170F" Font-Size="0.9em"/>
<TodayDayStyle BackColor="#F4000A" ForeColor="White" Font-Size="0.8em"
```

```
    Font-Bold="True"/>
<NextPrevStyle VerticalAlign="Bottom" Font-Bold="True" ForeColor="White"
    Font-Size="0.8em"/>
<DayHeaderStyle Font-Bold="True" Font-Size="0.8em" BackColor="#F4000A"
    ForeColor="White"/>
<TitleStyle BackColor="#8A170F" BorderColor="Black" Font-Bold="True"
    ForeColor="White" Font-Size="0.9em"/>
    <DayStyle Font-Size="0.8em" />
</asp:Calendar>
```

位于 Blue 主题文件夹中的 Calendar.skin 文件源代码如下：

```
<asp:Calendar runat="server" BackColor="White" BorderColor="#EFE6F7"
    CellPadding="4" DayNameFormat="Shortest" Font-Size="0.8em"
    ForeColor="Black" Height="180px" Width="200px">
<SelectedDayStyle BackColor="#41519A" Font-Bold="True" ForeColor="White"
    Font-Size="0.8em"/>
    <SelectorStyle BackColor="#41519A" Font-Size="0.8em"/>
    <WeekendDayStyle BackColor="#99ACDD" Font-Size="0.8em"/>
    <OtherMonthDayStyle ForeColor="#41519A" Font-Size="0.9em"/>
    <TodayDayStyle BackColor="#B2C3E1" ForeColor="Black" Font-Size="0.8em"/>
<NextPrevStyle VerticalAlign="Bottom" Font-Bold="True" ForeColor="White"
    Font-Size="0.8em"/>
    <DayHeaderStyle Font-Bold="True" Font-Size="0.8em" BackColor="#B2C3E1"/>
<TitleStyle BackColor="#41519A" BorderColor="Black" Font-Bold="True"
    ForeColor="White" Font-Size="0.9em"/>
```



```
    <DayStyle Font-Size="0.8em" />
</asp:Calendar>
```

如上所示，两段代码都用于设置日历控件的皮肤。设置的内容主要是一些样式属性，但所设置的属性值不相同。当以上两个控件皮肤随同主题应用到日历控件上时，将显示如图 7-5 和图 7-6 所示的外观。

下面列举了 Default.aspx 文件源代码。

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>


<script runat="server">
    void Page_PreInit(Object sender, EventArgs e)
    {
        //设置页面所设置的主题
        string theme = "";
        if (Request.QueryString["theme"] == null)
        {
            theme = "Red";
        }
        else
        {
            theme = Request.QueryString["theme"];
        }
        Page.Theme = theme;
        //设置 DropDownList 控件的选中项
        ListItem item = DropDownList1.Items.FindByValue(theme);
        if (item != null)
```

```
            {
                item.Selected = true;
            }
        }
        void SelectedIndexChanged(Object sender, EventArgs e)
        {
            //获取 DropDownList 选中项值，并进行页面重定向
            string url = Request.Path + "?theme=" + DropDownList1.SelectedItem.Value;
            Response.Redirect(url);
        }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>示例</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <fieldset style="width: 210px">
                <legend>动态加载主题</legend>
                <asp:DropDownList ID="DropDownList1" runat="server"
                        OnSelectedIndexChanged="SelectedIndexChanged"
                    AutoPostBack="True">
                    <asp:ListItem Value="Red">启用 Red 主题</asp:ListItem>
                    <asp:ListItem Value="Blue">启用 Blue 主题</asp:ListItem>
```



```
                </asp:DropDownList>
                <br />
                <asp:Calendar ID="Calendar1" runat="server"></asp:Calendar>
            </fieldset>
        </div>
    </form>
</body>
</html>
```

如上代码所示，页面中除包含 Calendar 和 DropDownList 控件之外，还包括了一段脚本代码，代码中的 Calendar 控件，只是进行了最简单的定义，没有任何与样式或其他有关的属性设置。当页面加载主题后，前文设置的控件皮肤将自动格式化该控件。DropDownList 控件中包含两个选项，一个是"启用 Red 主题"项，其 Value 为 Red，另一个是"启用 Blue 主题"项，其 Value 为 Blue。用户可将该控件的选中项 Value 作为页面的主题进行动态加载。脚本代码主要实现获取 DropDownList 控件选中项值，以及根据该值，通过 Page.Theme 设置页面主题。需要注意的是，Theme 属性设置必须在 Page_PreInit 事件处理中完成。然而，由于该事件在页面执行周期中引发过早，以至于无法使用 DropDownList 控件的任何属性。因此，代码中采用了将选中项值存储在 URL 的 QueryString 中的方法。

【小结】

- 了解网站开发的一般步骤。
- 创建主题。
- 应用主题。

【自测题】

1. 主题由一个文件组构成，其中可以包括(　　)。

A. 皮肤文件　　B. CSS 文件

C. 图片文件　　D. .aspx 文件

2. 皮肤文件的扩展名是(　　)。

A. .css　　B. .master　　C. .skin　　D. .aspx

3. 一般在(　　)事件中动态设置主题。

A. PreInit　　B. Init　　C. Load　　D. Render

【上机部分】

上机目标

- 创建主题
- 应用主题

上机练习

◆ 第一阶段 ◆

练习：创建主题外观文件。

【问题描述】

本示例以一个留言板为例，介绍创建外观文件方案的实施过程。通过对这 3 个 TextBox 控件的外观设置，可以看出应用主题可创建出美观的界面。运行效果如图 7-7 所示。



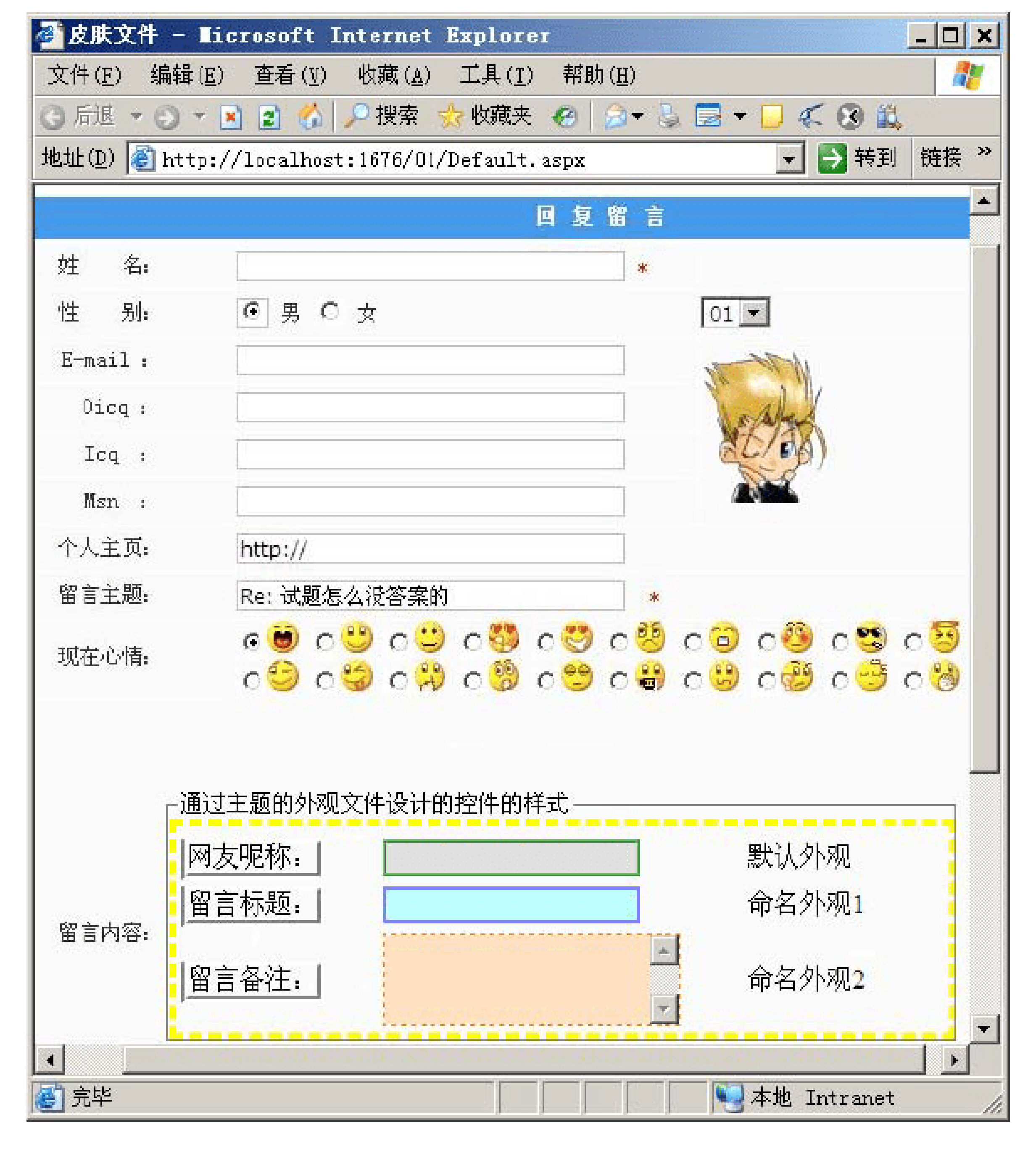


图 7-7

【问题分析】

在该示例中主要包括 3 个 TextBox 控件，其中一个设置了默认外观，另外两个设置了命名外观。

【参考步骤】

(1) 启动 Visual Studio 2008，新建一个网站。

(2) 在该网站的解决方案下，右键单击网站名称，选择“新建文件夹”命令，创建一个名为 App_Themes 的文件夹。同样步骤在该文件夹下再创建一个名为 mytheme 子文件夹，此时右键单击该子文件夹 mytheme，选择“添加新项”命令，将弹出“添加新项”对话框，如图 7-8 所示。

(3) 在“模板”列表中，选中“外观文件”图标，并在“名称”文本框中将其命名为“TextBox.skin”，单击“添加”按钮。此时外观文件将保存在 App_Themes 文件夹下的子文件夹 mytheme 中，文件组织示意图如图 7-9 所示。

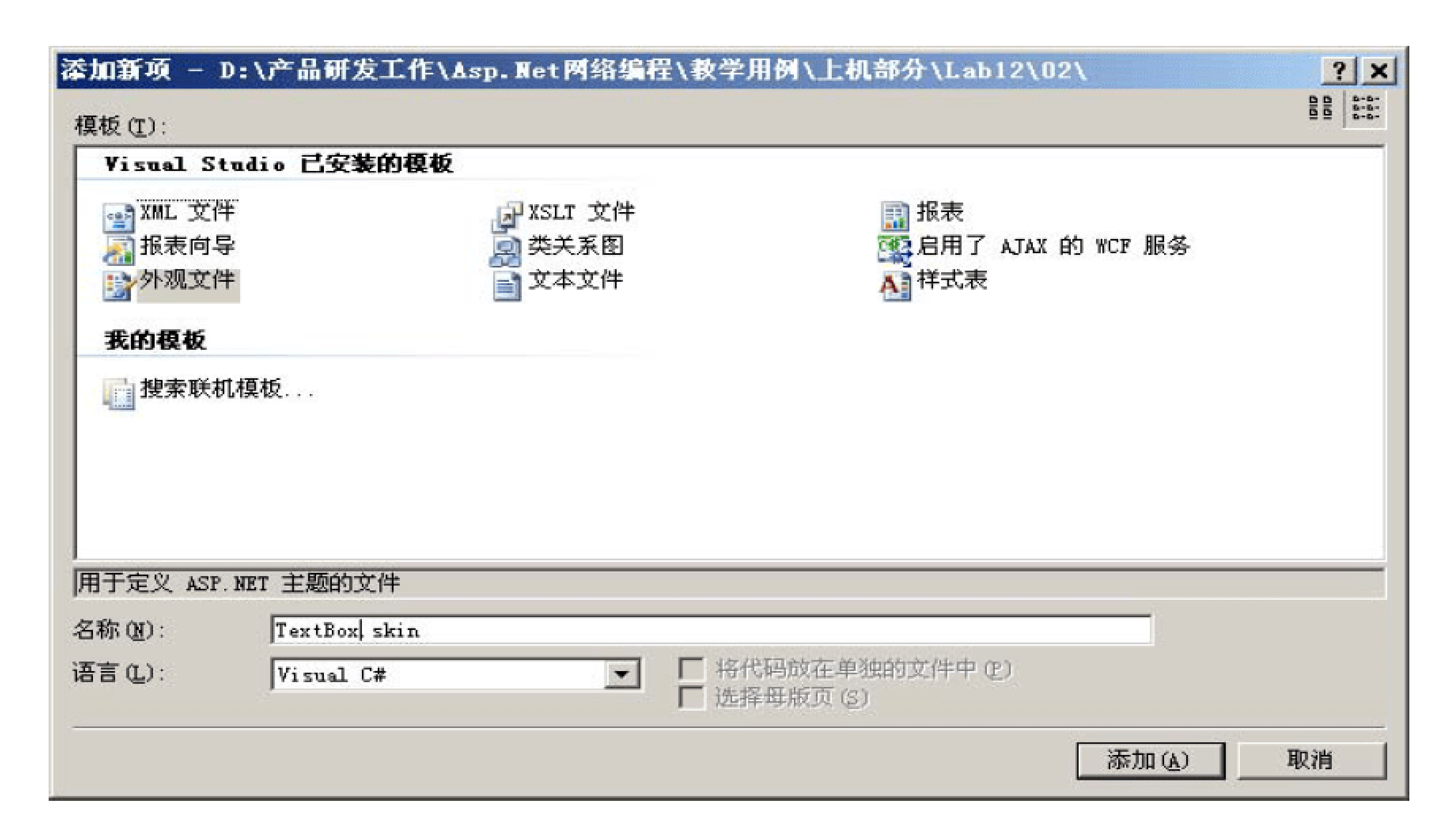


图 7-8



图 7-9

外观文件 TextBox.skin 中包含了对 TextBox 控件的外观设置。文件源代码如下：

```
<asp:TextBox  runat="server" BackColor="#E0E0E0" BorderColor="#80FF80"
BorderStyle="Ridge" ></asp:TextBox>


<asp:TextBox  runat="server" BackColor="#C0FFFF" BorderColor="#8080FF" BorderStyle="Solid"
SkinID="bule"></asp:TextBox>


<asp:TextBox  runat="server" BackColor="#FFE0C0" BorderColor="#FF8000" BorderStyle="Dotted"
SkinID="yellow"></asp:TextBox>
```



在以上的代码中包含 SkinID 属性的 TextBox 控件将拥有命名外观，而没有添加 SkinID 属性的 TextBox 控件将被设置为默认外观。

(4) 创建一个 Default.aspx 页面，并在该页面上添加 3 个 TextBox 控件，并应用 TextBox.skin 文件中的控件外观设置。该页面的源代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true" Theme="mytheme" CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>皮肤文件</title>
    <style type="text/css">
        .style1
        {
            height: 284px;
        }
        .style2
        {
            height: 295px;
        }
        .style3
        {
            height: 295px;
            width: 104px;
        }
        .style4
        {
```

```
                height: 284px;
                width: 104px;
            }
            .style5
            {
                width: 104px;
            }
        </style>
</head>
<body>
        <form id="form1" runat="server">
        <div>
            <table style="background-image: url('images/liuyanban%20.jpg');
                    width: 603px; height: 721px">
                <tr><td class="style3"> </td><td class="style2"></td></tr><tr>
                <td class="style4"> </td><td valign="top" class="style1">
                        <fieldset style="width: 240px">
                    <legend class="mainTitle"><span style="font-size: 10pt">
                    通过主题的外观文件设计的控件的样式 </span></legend>
                    <table style="border-top-style: dashed; border-right-style:
                            dashed; border-left-style: dashed; border-bottom-style: dashed;
                            border-left-color: #ffff00; border-bottom-color: #ffff00; width:
                            481px; border-top-color: #ffff00; border-right-color: #ffff00;">
                <tr>
                    <td colspan="2">
                        <span style="font-size: 11pt; border-top-style: inset;
```



```
            border-right-style: inset; border-left-style: inset;
            border-bottom-style: inset;"></span></td>
          <td colspan="1">
          </td>
      </tr>
      <tr>


          <td style="width: 100px">
              <span style="font-size: 11pt; border-top-style: groove;
                      border-right-style: groove; border-left-style: groove;
                      border-bottom-style: groove;">网友昵称：</span></td>
          <td style="width: 100px">
              <asp:TextBox ID="TextBox1" runat="server" ></asp:TextBox>
              </td>
          <td style="width: 100px">
              <span style="font-size: 11pt">默认外观</span></td>
      </tr>
      <tr>
          <td style="width: 100px">
              <span style="font-size: 11pt; border-top-style: groove;
                        border-right-style: groove; border-left-style: groove;
                        border-bottom-style: groove;">留言标题：</span></td>
          <td style="width: 100px">
              <asp:TextBox ID="TextBox2" runat="server" SkinID="bule" />
</td>
          <td style="width: 100px">
```

```
                        <span style="font-size: 11pt">命名外观 1</span></td>
                </tr>
                <tr>
                    <td style="width: 100px">
                        <span style="font-size: 11pt; border-top-style: groove;
                                border-right-style: groove; border-left-style: groove;
                                border-bottom-style: groove;">
                            留言备注：</span></td>
                    <td style="width: 100px">
                        <asp:TextBox ID="TextBox3" runat="server"
                                TextMode="MultiLine" SkinID="yellow" Rows="3"
                                BorderWidth="1px"></asp:TextBox></td>
                    <td style="width: 100px">
                        <span style="font-size: 11pt">命名外观 2</span></td>
                </tr>
            </table>
            </fieldset></td></tr>
                <tr><td class="style5"> </td><td></td></tr>
            </table>
    </div>
    </form>
</body>
</html>
```

SkinID 属性是创建和应用控件命名外观设置的关键，另外，上述代码中在 ID 为“TextBox3”的 TextBox 控件中还定义了 TextMode 和 Rows 属性，这说明控件的外观设置并不会独占控件，控件外观和页中的控件属性设置将合并，并构成控件的最终外观。



◆ 第二阶段 ◆

练习：为主题添加 CSS 样式。

【问题描述】

本阶段练习为主题添加 CSS 样式方案的实施过程，在本例中网站的功能区应用了主题 CSS 样式，使得整个网站色调看起来协调、美观。示例效果如图 7-10 所示。



图 7-10

【问题分析】

界面的搜索和导航功能区中共有 3 处设置了主题的 CSS 样式，一是页面背景颜色，二是超文本的外观及悬停效果，三是普通 HTML 文本框的背景色。由于此功能区中没有使用服务器控件，所以不能使用控件外观文件对其进行外观设置，只能使用 CSS。

【参考步骤】

(1) 打开 Visual Studio 2008，新建一个网站。

(2) 在该网站的解决方案下，右键单击网站名称，选择“新建文件夹”命令，创建一个名为 App_Themes 的文件夹。同样步骤在该文件夹下再创建一个名为 mytheme 的子文件夹，此时右键单击该子文件夹 mytheme，选择“添加新项”命令，将弹出“添加新项”对话框，如图 7-11 所示。

添加新项 - D:\产品研发工作\Asp.Net网络编程\教学用例\上机部分\Lab12\02\

模板(T):

Visual Studio 已安装的模板

XML 文件	XSLT 文件	报表
报表向导	类关系图	启用了 AJAX 的 WCF 服务
外观文件	文本文件	样式表

我的模板

搜索联机模板...

用于 Rich HTML 样式定义的级联样式表

名称(N): StyleSheet.css

语言(L): Visual C#　　将代码放在单独的文件中(P)　　选择母版页(S)

添加(A)　取消

图 7-11

(3) 在“模板”列表中，选中“样式表”图标，并在“名称”文本框中将其命名为“StyleSheet.css”，单击“添加”按钮。此时外观文件将保存在 App_Themes 文件夹下的子文件夹 mytheme 中。StyleSheet.css 文件的源代码如下:

```
body
{
    background-color: silver;
}

A:link
{
    color: Blue;
```



```
    text-decoration: underline;
}
A:visited
{
    color: Blue;
    text-decoration: underline;
}
A:hover
{
    color: Red;
    text-decoration: none;
}
INPUT
{
    background-color: Orange;
}
```

以上代码中，包含了 5 个样式表元素，它们分别用于设置页面的元素 body、用于设置超级链接文本的 A.link、A.visited、A.hover，以及设置输入控件 INPUT。可以看出，主题中包含的 CSS 文件与普通 CSS 文件没有任何区别。

(4) 创建一个 Default.aspx 页面，将创建的 StyleSheet.css 文件应用到这个页面和控件的皮肤设置中。Default.aspx 文件的源代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true" Theme="mytheme" CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head runat="server">
    <title>为主题添加 CSS 样式</title>
    <style type="text/css">
        .style1
        {
            height: 58px;
        }
        .style2
        {
            height: 58px;
            width: 437px;
        }
        .style4
        {
            height: 58px;
            width: 68px;
        }
        .style6
        {
            height: 58px;
            width: 322px;
        }
        .style8
        {
            height: 58px;
            width: 72px;
```



```
        }
        .style9
        {
            height: 481px;
            width: 322px;
        }
        .style10
        {
            height: 481px;
            width: 437px;
        }
        .style11
        {
            height: 481px;
            width: 68px;
        }
        .style12
        {
            height: 481px;
            width: 72px;
        }
        .style13
        {
            height: 481px;
        }
    </style>
```

```
</head>
<body>
    <form id="form1" runat="server">
        <table style="background-image: url('image/mr-demo.bmp');height: 518px;" >
            <tr class="style1">
<td class="style6"> </td><td valign="bottom" class="style2">
                    <input type="text" style="width: 404px; margin-bottom: 0px;" />
</td>
<td class="style4"></td>
                <td valign="bottom" class="style8"><span style="font-size: 11pt">
<span style="color: #0000ff; text-decoration: underline">
                    高级搜索</span>
</td>
<td valign="bottom" class="style1">
<a href="Default.aspx" target="_blank">
<span style="font-size: 11pt">发布资源</span></a>
</td>
</tr>
                <tr valign="top">
<td class="style9"></td>
<td class="style10"></td>
                    <td class="style11"></td>
                    <td class="style12"></td><td class="style13"></td>
</tr>
        </table>
    </form>
```



```
</body>
</html>
```



注意

应用主题中的 CSS 样式，需要在<%@ Page %>标签中设置 Theme 属性值为主题名；必须保证在页面头部定义<head runat="server">，否则主题中的 CSS 文件将无法正常应用。

【课后作业】

1. 修改上机部分第一阶段的练习，实现在应用程序级启用主题和禁用主题的功能。
2. 修改上机部分第一阶段的练习，实现在文件夹级启用主题和禁用主题的功能。

参 考 文 献

[1] 伊夫杰，等.ASP.NET 3.5 高级编程[M]. 北京：清华大学出版社，2008.

[2] 王岩. 精通 ASP.NET 3.5 企业级开发[M]. 北京：人民邮电出版社，2008.

[3] 郝刚，等. ASP.NET 2.0 开发指南[M]. 北京：人民邮电出版社，2006.